OPTOTERMINAL™
QLARITY FOUNDRY™
USER'S MANUAL
REVISION 1.5

OPTO 22
43044 Business Park Drive
Temecula, CA 92590-3614
USA

Phone: 800.OPTO (6786) or 951.695.3000
Fax: 800.832.OPTO (6786) or 951.695.2712
Email: sales@opto22.com
www.opto22.com

Manual 0059-01 (Opto 22 form 1344-030604)
3205E1 - Printed in USA

© Copyright QSI Corporation 2003

QTERM-G70, G70, QTERM-G75, G75, QTERM, Qlarity, and Qlarity Foundry are trademarks of QSI Corporation. OptoTerminal is a trademark of Opto 22. Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Win32, and their respective logos are registered trademarks of Microsoft Corporation in the United States and other countries.

The information in this manual has been checked carefully and is believed to be accurate; however, QSI Corporation and Opto 22 assume no responsibility for possible inaccuracies or omissions. Specifications are subject to change without notice.

FOREWORD

Qlarity Foundry™ is a Microsoft® Windows®-based design tool that uses the Qlarity™ programming language to help you design user applications for a Qlarity-based terminal such as the Opto 22 OptoTerminal.

Chapter 1 **Introduction.** This chapter explains how to use this manual to get the most out of Qlarity Foundry and describes Qlarity Foundry.

Chapter 2 **Installation.** This chapter covers the installation and setup of the Qlarity Foundry software.

Chapter 3 **Getting Started.** This chapter outlines Qlarity Foundry's features and functions.

Chapter 4 **Workspaces.** This chapter explains how to start, open, close, save, and use workspaces in Qlarity Foundry.

Chapter 5 **Templates, Resources and Libraries.** This chapter explains how to add and edit templates (advanced function), edit resources, and edit libraries.

Chapter 6 **Qlarity Foundry Preferences.** This chapter explains how to set up and define your terminal settings for Qlarity Foundry and enter Qlarity Foundry preferences.

Chapter 7 **Download Software to the Terminal.** This chapter explains how to configure the terminal communications port, download user applications and BFF files, and upgrade new firmware.

Chapter 8 **Basic Design.** This chapter explains how to get started using Qlarity Foundry and covers user application design basics.

Chapter 9 **Intermediate Design.** This chapter provides instructions a step beyond design basics.

Chapter 10 **Advanced Design.** This chapter provides instructions for advanced user application design.

Appendix A **Glossary of Software Terms.** This appendix provides definitions of the terms used in Qlarity Foundry.

Appendix B **EZObjects**. This appendix describes and explains how to use the objects in the Qlarity Foundry EZObjects library. EZObjects contain the programming needed to respond to an event.

MANUAL CONVENTIONS

Within Qlarity Foundry, functions can be selected using a mouse, a keyboard shortcut, or an icon on a toolbar. The following conventions are used to identify selections in this manual.

When instructed to press a specific key, it is shown in a bold, sans serif typeface and is enclosed in angle brackets:	**<Enter>**
When instructed to press a key and hold it down while typing another key, the keys are shown together:	**<CTRL>+<V>**
When instructed to click a button or icon, it is shown in a bold, sans serif typeface and is enclosed in square brackets:	**[OK]**
Icons on toolbars are represented graphically:	Click on the toolbar.
When instructed to type a character or a word, it is shown in the Courier typeface:	`e:\setup`
Directory paths, file names, and file name extensions are indicated by italics:	*eventbuilder.qly*
Syntax, commands, and examples are shown in the Courier typeface:	`dim count as integer`

CONTENTS

CHAPTER 1.
INTRODUCTION 1
1.1 How to Use This Manual 1
1.2 Description 1

CHAPTER 2.
INSTALLATION 3
2.1 System Requirements 3
2.2 Install Qlarity Foundry 3

CHAPTER 3.
GETTING STARTED 5
3.1 Start Qlarity Foundry 5
3.2 Open a Workspace 5
3.3 Main Window 6
3.4 Menu Options 7
3.4.1 File Menu 7
3.4.2 Edit Menu 8
3.4.3 View Menu 8
3.4.4 Tools Menu (Layout View only) 9
3.4.5 Help Menu 9
3.5 Toolbar 10
3.5.1 Layout View Toolbar 10
3.5.2 Code View Toolbar 10
3.6 Miscellaneous Bar 11
3.7 Object Tree 11
3.7.1 Globals 12
3.7.2 Object Templates 12
3.8 Layout and Code Views 12
3.9 Simulation 13
3.9.1 Serial I/O Support 14
3.9.2 Simulation Limitations 14
3.10 Object Properties 14
3.11 Object Palette 15
3.11.1 Add a New Object Instance 15
3.12 Move and Resize Windows 16
3.13 Where to Go From Here 16

CHAPTER 4.
WORKSPACES 17
4.1 Start a New Workspace 17
4.2 Open a Workspace 18
4.3 Close a Workspace 18
4.4 Save a Workspace 18
4.4.1 Save Workspace 19
4.4.2 Save Workspace As 19
4.4.3 Collect for Output 19
4.5 Compile a Workspace 19
4.6 Generate a BFF File 19

CHAPTER 5.
TEMPLATES, RESOURCES, AND LIBRARIES 21
5.1 Add/Edit Templates 21
5.1.1 Add a New Object Template 21
5.1.2 Edit Icon 22
5.1.3 Rename Template 22
5.1.4 Remove Template 22
5.1.5 Send Template to Library 23
5.2 Edit Resources 23
5.2.1 Add a Resource 24
5.2.2 Preview Resources 24
5.2.3 Rename a Resource 25
5.2.4 Change a Resource File 25
5.2.5 Remove a Resource 25
5.2.6 Bitmaps 25
5.2.7 Fonts 25
5.2.8 Audio 26
5.2.9 Binary 26
5.3 Edit Libraries 26
5.3.1 Add Existing Library 27
5.3.2 Remove Library 27
5.3.3 Edit Library 27
5.3.3.1 Edit Entry 28
5.3.3.2 Rename Entry 28
5.3.3.3 Remove Entry 28
5.3.3.4 Set Entry Version 28
5.3.3.5 Add New Entry 28
5.3.4 Edit Natives 29
5.3.5 Create a New Library 29

CHAPTER 6.
QLARITY FOUNDRY PREFERENCES 31
6.1 Terminal 31
6.1.1 Display Setup 31
6.1.2 Input 31
6.2 Layout 31
6.3 Editor 33
6.3.1 Font 33
6.3.2 Colors 33
6.4 Compile 33
6.5 Simulation 34
6.5.1 Serial Port Setup 35
6.5.2 Communications Window Settings 35
6.5.3 Keypad Settings 36

CHAPTER 7.
DOWNLOAD SOFTWARE TO THE TERMINAL 37
7.1 Configure Communications Port 37
7.1.1 Serial Port Settings 37
7.1.2 Ethernet Port Settings 37
7.2 Download a User Application 38

7.2.1 Prepare the Terminal for Downloading 38
7.2.2 Download the User Application 38
7.3 Download a BFF File 38
7.4 Upgrade the Firmware 38
7.4.1 Determine Current Firmware Version 39
7.4.2 Prepare Terminal for Upgrade 39
7.4.3 Download New Firmware 39
7.4.4 Induce Bootloader 40
7.4.5 Set Unit Time 40

CHAPTER 8.
BASIC DESIGN 43
8.1 Prepare Qlarity Foundry for Application Design 43
8.1.1 Basic Design Layout 43
8.1.2 Simulate the Terminal Display 44
8.1.3 Drawing Aids 45
8.1.4 Add/Remove Resources 45
8.1.5 Add/Remove Libraries 46
8.1.5.1 Libraries Provided with Qlarity Foundry 46
8.2 Understanding Qlarity for Basic Design 47
8.2.1 Workspaces and User Applications 47
8.2.2 Qlarity Objects 47
8.2.3 Parent/Child Relationships 47
8.2.4 Z-Order 48
8.2.5 Events and Messaging 48
8.2.6 Enabled/Disabled Objects 49
8.3 Design a User Application 49
8.3.1 Add an Object Instance 49
8.3.1.1 Add an Object From the Object Palette 49
8.3.1.2 Add an Object From the Shortcut Menu 49
8.3.2 Move, Resize and Reorder Objects 50
8.3.2.1 Move an Object 50
8.3.2.2 Resize an Object 50
8.3.2.3 Change the Order of Objects 50
8.3.3 Change Object Properties 51
8.3.3.1 Select Color 52
8.4 Event Builder 53
8.4.1 Overview of Event Builder Steps 53
8.4.2 Event Builder Dialog Box 53
8.4.3 Select and Configure Actions 54
8.4.3.1 Select Actions 54
8.4.3.2 Configure Actions 55
8.4.4 Load Event Builder Sample Workspace 57
8.4.4.1 Tank Demo 57
8.4.4.2 Toggle Demo 57
8.4.4.3 Keypad Demo 57
8.4.5 Qlarity Code and Event Builder 57
8.4.6 Troubleshooting 58
8.5 Communication Objects 58
8.5.1 Serial Objects 58
8.5.2 Ethernet Objects 58

8.5.3 Receive Data ... 59
8.6 Test the User Application ... 59
8.7 Save and Compile a Workspace ... 60
8.7.1 Save a Workspace ... 60
8.7.2 Compile a Workspace ... 60
8.8 Download a User Application ... 60

CHAPTER 9.
INTERMEDIATE DESIGN ... 61
9.1 Viewing the Code ... 61
9.2 Understanding Qlarity for Intermediate Design ... 62
9.2.1 Qlarity Programming Language ... 62
9.2.2 Objects and Templates ... 62
9.3 Qlarity Code for Objects ... 63
9.3.1 Property Initializations ... 63
9.3.2 Method Overrides ... 63
9.4 Programmable Objects Library ... 63
9.4.1 Override an Object Method ... 64
9.5 Global Code ... 65
9.5.1 Add a Global Variable to a Workspace ... 65
9.5.2 Add a Global Function to a Workspace ... 65
9.5.3 Add a Global Event Handler to a Workspace ... 66
9.6 Create a New Object Template ... 66
9.7 Where to Go From Here ... 68

CHAPTER 10.
ADVANCED DESIGN ... 69
10.1 Validation Functions ... 69
10.2 The Qlarity API Library ... 70
10.3 Exception Handling ... 70
10.4 Create a New Object Template ... 71
10.4.1 New Template Boilerplate Code ... 72
10.4.1.1 Non-Drawable Objects ... 74
10.4.1.2 Area Objects ... 74
10.4.1.3 Container Objects ... 75
10.4.2 Getting New Object Templates to Work in Qlarity Foundry ... 75
10.4.3 Adding Object Template Documentation ... 79
10.5 Guidelines for Designing New Object Templates ... 80
10.6 Where to Go From Here ... 81

APPENDIX A.
GLOSSARY OF SOFTWARE TERMS ... 83

APPENDIX B.
EZOBJECTS LIBRARY ... 85
B.1 Set Up an Event with EZOBjects ... 85

CHAPTER 1

INTRODUCTION

1.1 How to Use This Manual

For instructions on installing Qlarity Foundry™, see Chapter 2, "Installation."

To get started using Qlarity Foundry and to learn design basics, review the following chapters:

- ❒ Chapter 3, "Getting Started"
- ❒ Chapter 8, "Basic Design"

To learn more about user application design, refer to the following chapters:

- ❒ Chapter 9, "Intermediate Design"
- ❒ Chapter 10, "Advanced Design"

Use the following chapters to learn about specific Qlarity Foundry functions:

- ❒ Chapter 4, "Workspaces"
  How to start, open, close, save, and use workspaces in Qlarity Foundry.
- ❒ Chapter 5, "Templates, Resources, and Libraries"
  How to add and edit templates (advanced function), edit resources, and edit libraries.
- ❒ Chapter 6, "Qlarity Foundry Preferences"
  How to set up and define your terminal settings for Qlarity Foundry and enter Qlarity Foundry preferences.
- ❒ Chapter 7, "Download Software to the Terminal"
  How to configure the terminal communications port, download user applications and BFF files, and upgrade new firmware.
- ❒ Appendix A, "Glossary of Software Terms"
  A list of Qlarity™ terms and their definitions. You should read and understand these terms before you attempt to perform the functions described in Chapter 9 and Chapter 10.
- ❒ Appendix B, "EZObjects Library"
  The EZObjects library contains object templates that have "event" functionality programmed into them.

1.2 Description

Qlarity (pronounced "clarity"), the programming language used to program an Opto 22 OptoTerminal, is a powerful, BASIC-like language that utilizes the full potential of this family of terminals. *Qlarity Foundry* is a Microsoft® Windows®-based design tool that uses Qlarity to help you design user applications for these terminals.

Qlarity uses "objects" as the building blocks for all user applications. An object can take many forms, including text labels and fields, bitmaps, lines, forms, key definitions, clocks, counters, and so on.

Libraries, which are supplied with Qlarity Foundry, contain predefined objects for which the programming is already done. To adapt a library object to a user application, you typically only need to enter properties (e.g., name, location, color, bitmap, etc.). If custom programming is required, Qlarity Foundry contains the tools you need to modify library objects or create your own objects.

Qlarity Foundry provides all the tools you need to work with user applications, including those used to do the following:

- Create a user application (basic to advanced levels).
- Modify a user application.
- Compile and download a user application to the terminal.

CHAPTER 2

INSTALLATION

This chapter provides instructions for installing and setting up the Qlarity Foundry software.

2.1 System Requirements

Computer Requirements:

- Pentium 166 or better
- 32 Mbytes of RAM (64 Mbytes recommended)
- 30 Mbytes available hard disk space

Operating System Requirements:

Microsoft® Windows® 95, 98, 2000, NT (4.0 or later), or XP. Qlarity Foundry will not run in Windows 3.x or DOS. Windows 95 requires Internet Explorer 4.01 or higher. Windows NT requires Service Pack 6.

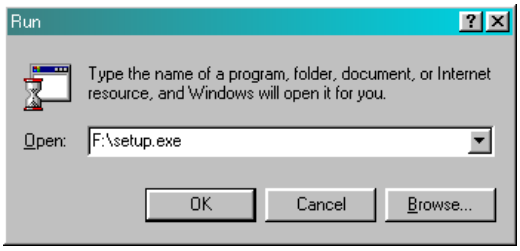
2.2 Install Qlarity Foundry

To install Qlarity Foundry, take the following steps.

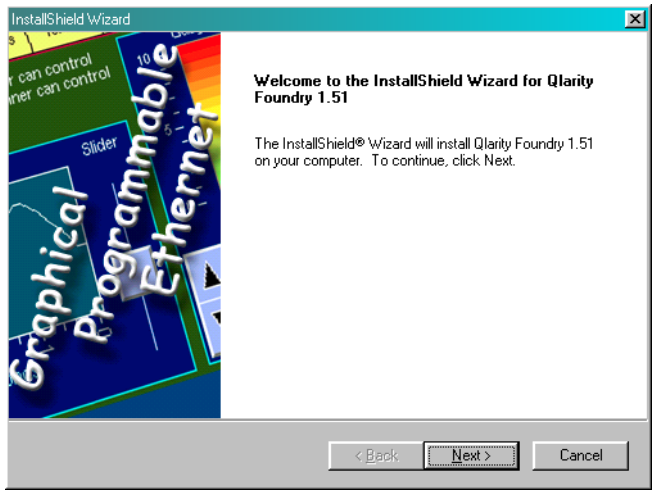
1. Close any open Windows applications.

2. Insert the Qlarity Foundry CD-ROM into your CD-ROM drive.

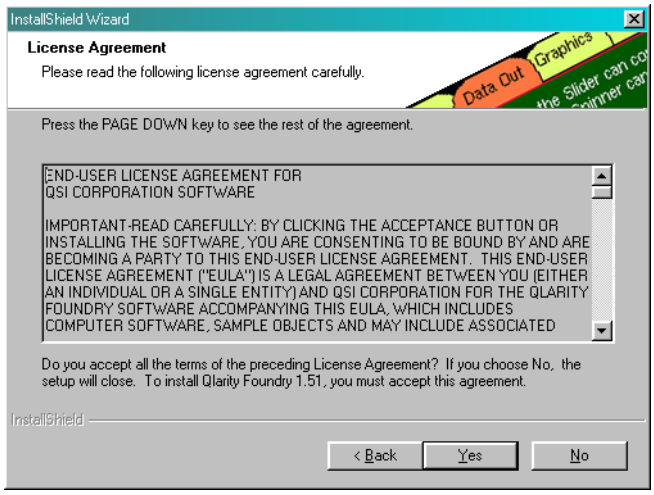
   The Setup program should auto-start. If it does not, click **[Start]**, and click **Run**. The following window appears.



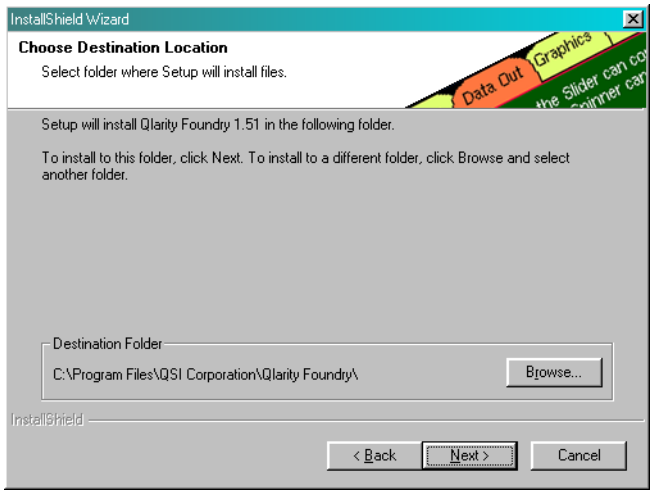
3. Type `e:\setup` (in which "e" is the letter of your CD-ROM drive). The following window appears.



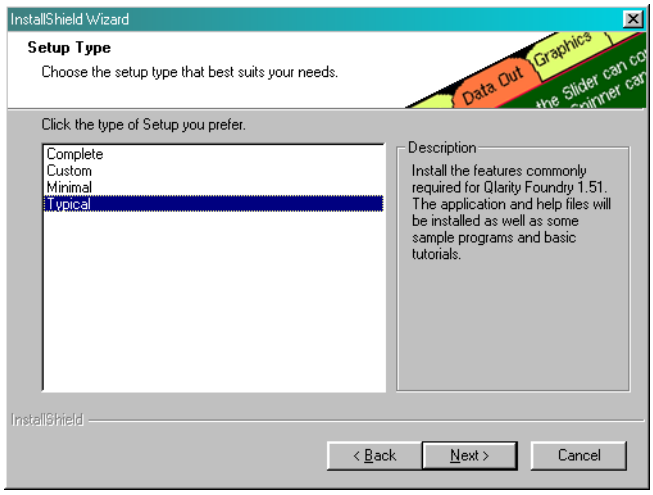
4. If no other Windows-based applications are running, click **[Next]**, and the License Agreement window is displayed.


InstallShield Wizard
License Agreement
Please read the following license agreement carefully.
Data Out
Graphics
Press the PAGE DOWN key to see the rest of the agreement.
END-USER LICENSE AGREEMENT FOR
QSI CORPORATION SOFTWARE
IMPORTANT-READ CAREFULLY: BY CLICKING THE ACCEPTANCE BUTTON OR INSTALLING THE SOFTWARE, YOU ARE CONSENTING TO BE BOUND BY AND ARE BECOMING A PARTY TO THIS END-USER LICENSE AGREEMENT. THIS END-USER LICENSE AGREEMENT ("EULA") IS A LEGAL AGREEMENT BETWEEN YOU (EITHER AN INDIVIDUAL OR A SINGLE ENTITY) AND QSI CORPORATION FOR THE QLARITY FOUNDRY SOFTWARE ACCOMPANYING THIS EULA, WHICH INCLUDES COMPUTER SOFTWARE, SAMPLE OBJECTS AND MAY INCLUDE ASSOCIATED
Do you accept all the terms of the preceding License Agreement? If you choose No, the setup will close. To install Qlarity Foundry 1.51, you must accept this agreement.
InstallShield
< Back
Yes
No

5. Read the agreement and click **[Yes]** if you accept the terms. The following window appears.

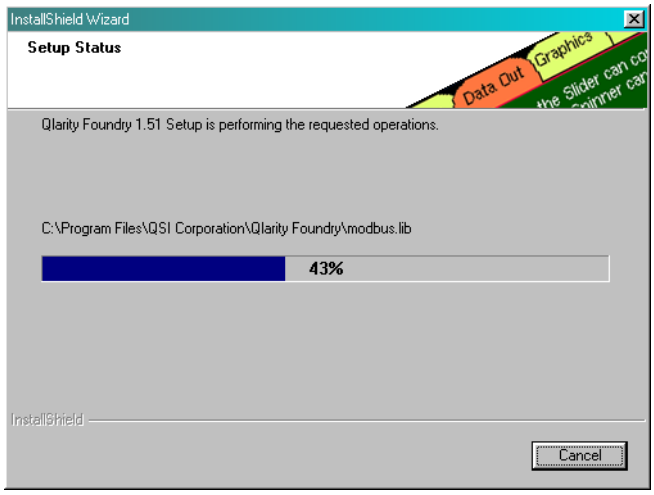

InstallShield Wizard
Choose Destination Location
Select folder where Setup will install files.
Data Out
Graphics
Setup will install Qlarity Foundry 1.51 in the following folder.
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.
Destination Folder
C:\Program Files\QSI Corporation\Qlarity Foundry\
Browse...
InstallShield
< Back
Next >
Cancel

6. Accept the destination folder shown, or click **[Browse]** to select a different folder or create a new folder in which to install Qlarity Foundry. Click **[Next]** to continue. The following window appears.

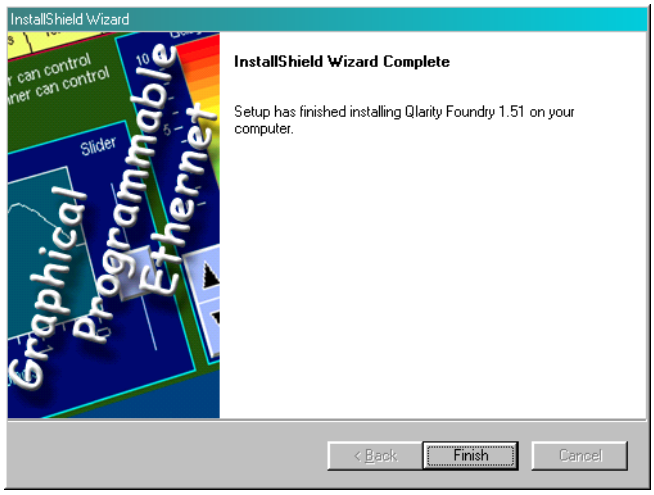

InstallShield Wizard
Setup Type
Choose the setup type that best suits your needs.
Data Out
Graphics
Click the type of Setup you prefer.
Complete
Custom
Minimal
Typical
Description
Install the features commonly required for Qlarity Foundry 1.51. The application and help files will be installed as well as some sample programs and basic tutorials.
InstallShield
< Back
Next >
Cancel

7. Click each type of system setup to read a description of the files that are installed. Then, select the type of setup that best suits your needs: **Complete**, **Custom**, **Minimal**, or **Typical**. Click **[Next]** to continue. If you selected "Custom," a dialog box appears for you to select the files and features you want to install.

   The Setup Status window appears and displays the status of the installation as each file is installed in the specified destination folder.



   After all of the files are copied to your hard disk, the following window appears.



8. Click **[Finish]**. Installation is complete.

CHAPTER 3

GETTING STARTED

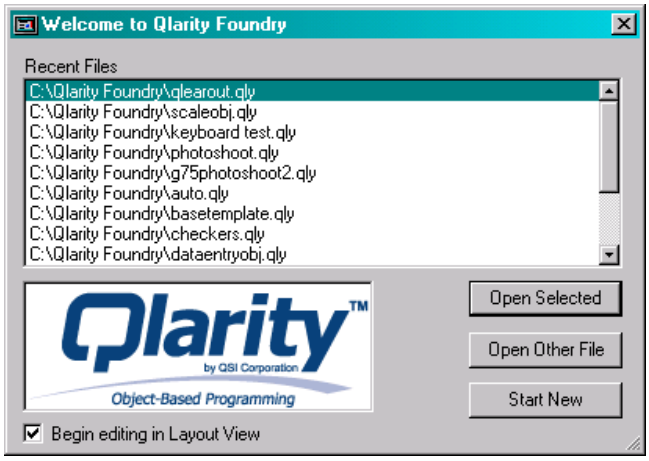
3.1 Start Qlarity Foundry

To start Qlarity Foundry, click **[Start]**, then click **Programs** and select **Opto 22 Software**. Click **Qlarity Foundry** to start the program.

3.2 Open a Workspace

A *workspace* is a file created in Qlarity Foundry that you will compile into a *user application*. In Qlarity Foundry, you use a workspace to define functions for the terminal; at the OptoTerminal, you use a user application to perform the functions.

When Qlarity Foundry starts, a dialog box appears so you can open a workspace.



Open Selected
Recently opened files, if any, are listed. To open a recent file, click the file name to select it, then click **[Open Selected]** (or double-click on the file name). The workspace is opened in the main window. Only one workspace at a time can be open in Qlarity Foundry.

Open Other File
Click **[Open Other File]** and the Open dialog box appears showing workspaces in the default folder. Use the Windows tools to change folders if necessary. Click a file to select it, then click **[Open].** The workspace is opened in the main window.

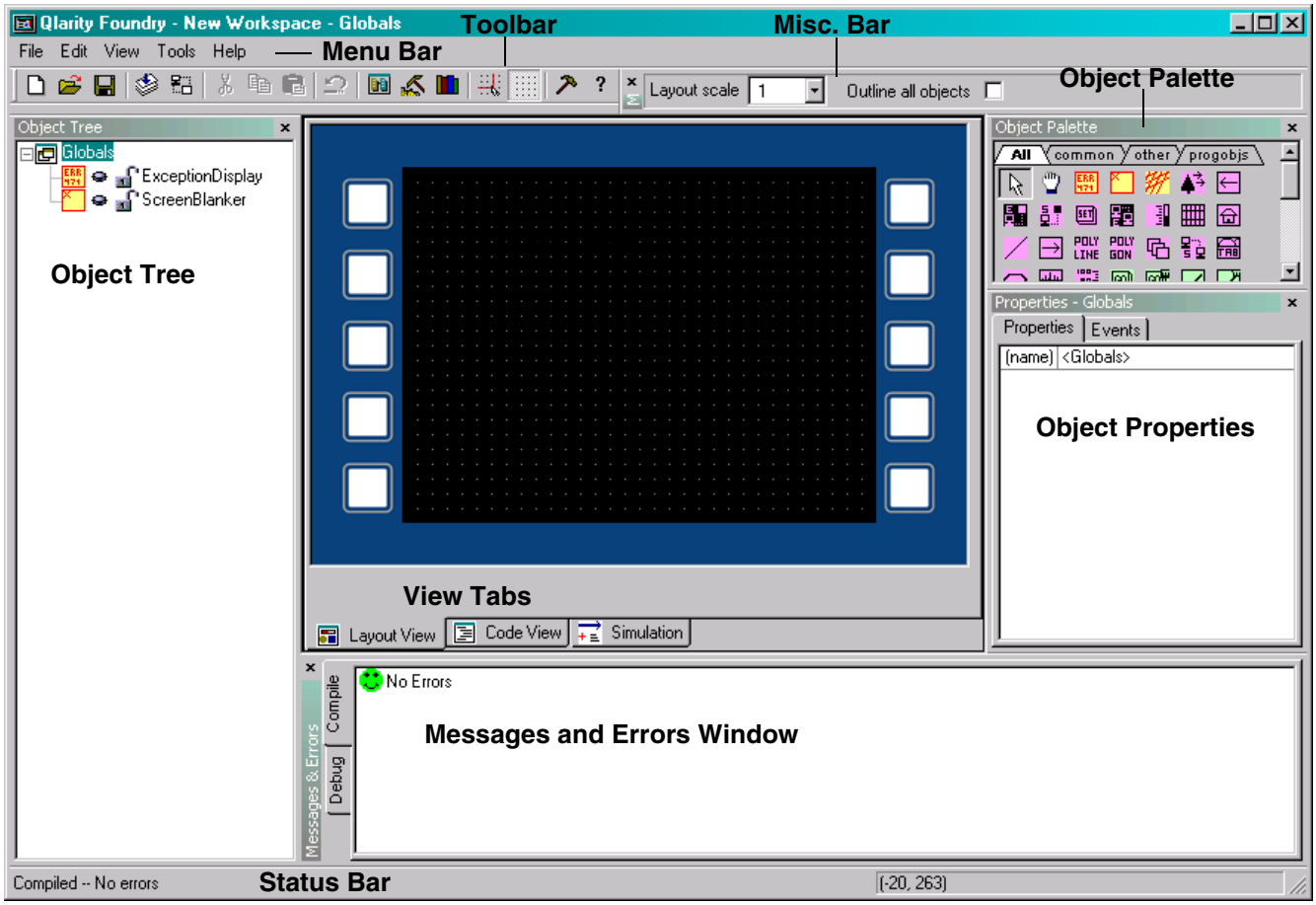
Start New
To start a new workspace, click **[Start New]**. The New Qlarity Project dialog box appears. Refer to section 4.1, "Start a New Workspace" for information.

Enable or disable the following option:

Begin editing in Layout View
If enabled, a workspace is automatically compiled and shown in the "Layout View" when it is opened. If disabled, the workspace is displayed in "Code View" and will need to be compiled before it can be displayed in Layout View. You can change this setting using the Settings function, Compile tab (see section 6.4, "Compile").

3.3 Main Window

The Qlarity Foundry main window is shown below.



Menu Bar
The menu bar contains the File, Edit, View, Tools and Help menus. Refer to section 3.4 for a brief description of the menu options. You can access all Qlarity Foundry options from the menus.

Toolbar
The toolbar contains tools that provide quick links to commonly used Qlarity Foundry functions. The tools available change depending on the selected view mode (Layout View or Code View). Refer to section 3.5 for a description of each tool on the toolbar.

Object Tree
Object instances that have been added to the workspace are listed under "Globals" and object templates are listed under "Templates" in the Object Tree. Some objects may be automatically added to a workspace when it is created, including the following:

- ScreenBlanker—blanks the screen after a period of inactivity
- ExceptionDisplay—displays unhandled Qlarity Exceptions

Work Area
This is the area in the Layout View in which you place object instances to add them to the workspace. The work area simulates the display of the OptoTerminal.

Object Properties
Object Properties is displayed by default. Click an object instance in the Object Tree or work area and its properties are displayed. Refer to section 3.10 for more information.

Object Palette
The Object Palette contains an icon for each object template in the workspace and in any active libraries. You click the icons to add new object instances. Move the mouse pointer over an icon to display the object's name. Refer to section 3.11 for more information.

Layout View/Code View/Simulation
To set up the workspace, you can work in either Layout View or Code View. The Simulation mode can be used after you have created a workspace to simulate the performance of the workspace objects on the OptoTerminal.

Click the Layout View tab or Code View tab at the bottom of the work area (default location) to change views. Click the Simulation tab to start the terminal simulation.

Layout View is a graphics mode used to lay out the terminal display. Generally, unless you are making changes to the programming code, you will work in Layout View.

Code View is used to write or edit an object's programming code. You will generally only work in Code View if you are customizing objects or creating your own objects.

Refer to section 3.8 for more information on the Layout and Code Views. Refer to section 3.9 for more information on Simulation mode.

Messages and Errors
The Messages and Errors window displays any errors or other messages generated when the workspace is compiled or during a "debug" operation. These messages can help you identify and correct problems with the workspace.

Status Bar
The status bar shows the status of the workspace and the coordinates at which the mouse pointer is positioned in the work area. Also, if you position the mouse pointer over an option on a menu or a tool on the toolbar, the function of the option or tool is displayed in the status bar.

3.4 Menu Options

The following sections describe the options available on each menu on the menu bar.

3.4.1 File Menu

New Workspace
Use this option to start a new workspace. Refer to section 4.1 for information.

Open Workspace
Use this option to open an existing workspace. Refer to section 4.2 for information.

Close Workspace
Select this option to close the current workspace and open a new workspace.

Save
Use this option to save the current workspace. Refer to section 4.4 for information.

Save Workspace As
Select this option to save a copy of the current workspace to a different file name. The Save As dialog box is displayed. Enter the new file name (the extension is added automatically), and click **[Save]**.

Collect for Output
Use this option to transfer a workspace to another computer See section 4.4.3 for information.

Compile
Use this option to compile the current workspace into the format required for a user application. A workspace must also be compiled to properly display in Layout View. Refer to section 4.5 for information on compiling a workspace.

Generate BFF File
You can download a user application that is not currently loaded in Qlarity Foundry to the terminal, but the file must be in binary file format (BFF). Use this option to save a file to BFF. Refer to section 4.6 for information.

Download Application
Select this option to download files to the terminal. You can download the currently loaded workspace, any BFF file, or a firmware upgrade. Refer to Chapter 7, "Download Software to the Terminal" for information.

Recent Files
After you have created and saved Qlarity files, you can use this option as a shortcut to open files that have been recently opened. Click Recent Files, and the five most recently opened files are listed. Click the file that you want to open.

Exit
Select this option to exit Qlarity Foundry.

3.4.2 Edit Menu

The options in the Edit menu change depending on whether you are in Layout View or Code View.

Cut
In Layout View, select an object in the work area, then select this option (or press <**Ctrl**>+<**X**>) to cut the object from the workspace. In Code View, select code, then select this option (or press <**Ctrl**>+<**X**>) to cut the selected code from the workspace. The object or code remains in the Windows buffer until replaced by another "cut" or "copy" operation.

Copy
In Layout View, select an object in the work area, then select this option (or press <**Ctrl**>+<**C**>) to make a copy of the object. In Code View, select code, then select this option (or press <**Ctrl**>+<**C**>) to copy the selected code. The object or code remains in the Windows buffer until replaced by another "cut" or "copy" operation.

Paste
In Layout View, select this option (or press <**Ctrl**>+<**V**>) to place a cut or copied object in a new location in the same workspace or in a different workspace. In Code View, select this option (or press <**Ctrll**>+<**V**>) to place cut or copied code at the cursor position in the same or a different workspace or in a different Windows application.

Undo
Select this option to undo the last action performed. In Layout View, the undo buffer is cleared each time the workspace is compiled. In Code View, the undo buffer is cleared each time you view the code of a different object.

Redo (Code View only)
Select this option to redo the most recent undone action. This option is only available after you have selected **Undo** in Code View.

Add/Edit Templates
Select this option to open the Add/Edit Templates dialog box to add or edit an object template. Refer to section 5.1, "Add/Edit Templates" and to Chapter 10, "Advanced Design" for information on creating templates.

Edit Resources
Use this option to open the Qlarity Resources dialog box to add resources to or remove them from the workspace. Refer to section 5.2, "Edit Resources" for information.

Edit Libraries
Use this option to open the Libraries dialog box to add libraries to or remove libraries from the workspace. Refer to section 5.3 for information.

Find (Code View only)
Select this option to find code in the displayed object instance, template or workspace. Enter the characters or code you want to find, and click **[Find Next]**. To find only characters matching the entered case (upper or lower), select **Match Case**. To search through all object instances and templates in the workspace, select **Entire Workspace**.

Replace (Code View only)
Select this option to find code in the displayed object instance or template and replace it with specified code. In the "Find What" field, enter the characters or code you want to find. In the "Replace With" field, enter the code with which you want to replace it. Click **[Find Next]** to find the code. Then click **[Replace]** to replace it. Or, click **[Replace All]** to automatically find and replace all occurrences of the code. To find only characters matching the entered case (upper or lower), select **Match Case**.

Goto Bookmark (Code View only)
Select this option to go to the next bookmark in the displayed object's code. Use Set Bookmark to add or remove bookmarks in an object's code.

Set Bookmark (Code View only)
Select this option to add or remove a bookmark in an object's code at the cursor location. Bookmarks make it possible to find a location quickly, which is useful in objects with several lines of code.

3.4.3 View Menu

Toolbar
Select this option to show or hide the toolbar.

Status Bar
Select this option to show or hide the status bar.

Misc. Bar
Select this option to show or hide the Layout Scale and Outline All Objects functions next to the toolbar.

Messages and Errors
Select this option to show or hide the Messages and Errors window, which displays any errors or other messages generated when the workspace is compiled or during a "debug" operation. These messages can help you identify and correct problems with the workspace.

Object Tree
Select this option to show or hide the Object Tree.

Object Palette (Layout View only)
Select this option to show or hide the Object Palette.

Object Properties (Layout View only)
Select this option to show or hide Object Properties.

Refresh (Layout View only)
Select this option to redraw the graphics on the screen in Layout View.

Layout View
Select this option to switch to the Layout View.

Code View
Select this option to switch to the Code View.

Simulation
Select this option to switch to the Simulation mode.

Settings
Use this option to open the Qlarity Foundry Preferences dialog box to define the configuration of the OptoTerminal and to enter Qlarity Foundry preferences. Qlarity Foundry attempts to simulate the terminal's display area, input type, and so on as closely as possible. Refer to Chapter 6, "Qlarity Foundry Preferences" for information.

3.4.4 Tools Menu (Layout View only)

View Only Enabled Objects
Disabled objects are displayed in Layout View by default. Select this option if you want to see only enabled objects in the work area.

NOTE: disabled objects vs. hidden objects
Keep in mind that the "enable/disable" option is different from the "show/hide" option. Disabled objects are not displayed on the terminal; hidden objects are displayed on the terminal (unless they are disabled), they are only hidden in Qlarity Foundry. Refer to section 3.7 for information on the "show/hide" option.

Stop Qlarity Interpreter
When a workspace is compiled, the Qlarity interpreter runs a portion of the user application in Qlarity Foundry. Select this option to stop the user application from running. For example, if an object instance contains an infinite loop, you will not be able to work in Layout View. You can select this option to terminate the interpreter and switch to Code View so you can correct the problem.

NOTE: error message
The Qlarity interpreter may detect the infinite loop when it tries to run the application and display a prompt for you to stop the application.

Snap to Grid
Select this option to enable or disable the "snap to grid" function. If enabled, this feature forces objects to "snap" to the nearest grid point when added or moved. The top left corner of an object's rectangle snaps to the nearest grid point when you release the mouse button.

View Grid
Select this option to show or hide the "grid." You can display a grid over the work area to help you more accurately place and align objects. For information on changing the spacing or color of the grid, refer to section 8.1.3, "Drawing Aids."

3.4.5 Help Menu

Show Object Documentation (F1)
Select this option, or press **<F1>,** to view Object Documentation, which is a Help file containing information on library and workspace object templates, as well as functions, variables, and APIs in the workspace. Documentation is included for all libraries provided with Qlarity Foundry. Advanced users can add documentation for object templates they create (refer to section 10.4.3, "Adding Object Template Documentation" for information).

To use Object Documentation, in the Qlarity Documentation index, double-click the item for which you want to view Help. You can also type a keyword in the text box to search for an item, then double-click the item in the keyword list.

Show Qlarity Foundry Help (Shift + F1)
Select this option, or press **<Shift>+<F1>,** to view the Qlarity Foundry Help file. The Help file provides the information in this Qlarity Foundry manual so that you can access it online while you are working in Qlarity Foundry.

Show Qlarity Language Help (Ctrl + F1)
Select this option, or press **<Ctrl>+<F1>,** to view the Programmer's Reference Help file. The Programmer's Reference provides information on the Qlarity programming language and message handling system, detailed information on the syntax to write programs in the Qlarity language, descriptions of system messages, a Qlarity API (Application Programming Interface) function reference, and other information on programming in the Qlarity language.

About Qlarity Foundry
Select this option to identify the version of Qlarity Foundry, BFF, and the firmware that you are running.

3.5 Toolbar

The Qlarity Foundry toolbar contains icons for commonly used functions. Click an icon to open the function. Select **Toolbar** from the View menu to show or hide the toolbar.

The following sections describe the tools available on the toolbar in Layout View and Code View.

3.5.1 Layout View Toolbar

Click to start a new workspace. Refer to section 4.1 for information.

Click to open an existing workspace. Refer to section 4.2 for information.

Click to save the current workspace. Refer to section 4.4 for information.

Click to compile the current workspace into the format required for a user application. A workspace also must be compiled to properly display in Layout View. Refer to section 4.5 for information on compiling a workspace.

Click to open the Download Application dialog box. Refer to Chapter 7, "Download Software to the Terminal" for information.

Select an object in the work area, then click to cut the object from the workspace. The object remains in the Windows buffer until replaced by another "cut" or "copy" operation.

Select an object in the work area, then click to make a copy of the object. The object remains in the Windows buffer until replaced by another "cut" or "copy" operation.

Click to place a cut or copied object in a new location in the same workspace or in a different workspace.

Click to undo the last action performed. (The Undo buffer is cleared after the workspace is compiled.)

Click to open the Add/Edit Templates dialog box to add or edit an object template. Refer to section 5.1, "Add/Edit Templates" and to Chapter 10, "Advanced Design" for information on creating templates.

Click to open the Qlarity Resources dialog box to add resources to or remove them from the workspace. Refer to section 5.2, "Edit Resources" for information.

Click to open the Libraries dialog box to add libraries to or remove libraries from the workspace. Refer to section 5.3 for information.

Click to turn on/off the "snap to grid" function. If enabled, this feature forces objects to "snap" to the nearest grid line when added or moved. The top left corner of an object's rectangle snaps to the nearest horizontal and vertical grid lines when you release the mouse button.

Click to show or hide the "grid." You can display a grid over the work area to help you more accurately place and align objects. For information on changing the spacing or color of the grid, refer to section 8.1.3, "Drawing Aids."

Click to open the Qlarity Foundry Preferences dialog box to define the terminal's configuration and to enter Qlarity Foundry preferences. Qlarity Foundry attempts to simulate the terminal's display area, input type, and so on as closely as possible. Refer to Chapter 6, "Qlarity Foundry Preferences" for information.

Click to identify the version of Qlarity Foundry you are running.

3.5.2 Code View Toolbar

Click to start a new workspace. Refer to section 4.1 for information.

Click to open an existing workspace. Refer to section 4.2 for information.

Click to save the current workspace. Refer to section 4.4 for information.

Click to compile the current workspace into the format required for a user application. A workspace also must be compiled to properly display in Layout View. Refer to section 4.5 for more information.

Click to open the Download Application dialog box. Refer to Chapter 7 for information.

Select code, then click to cut the selected code from the workspace. The code remains in the Windows buffer until replaced by other "cut" or "copied" code.

Select code, then click to copy the selected code. The code remains in the Windows buffer until replaced by other "cut" or "copy" operation.

Click to place cut or copied code into a workspace or another Windows application at the cursor position.

Click to go to the next bookmark in the displayed object. Use the Toggle Bookmark tool to add bookmarks to an object's code.

Click to add or remove a bookmark in an object's code at the cursor location. Bookmarks make it possible to find a location quickly, which is useful in objects with several lines of code.

Click to open the Add/Edit Templates dialog box to add or edit an object template. Refer to section 5.1, "Add/Edit Templates" and to Chapter 10, "Advanced Design" for information on creating templates.

Click to open the Edit Resources dialog box to add resources to, or remove them from, the workspace. Refer to section 5.2, "Edit Resources" for information.

Click to open the Libraries dialog box to add libraries to or remove libraries from the workspace. Refer to section 5.3 for information.

Click to open the Qlarity Foundry Preferences dialog box to define the terminal's configuration and to enter Qlarity Foundry preferences. Qlarity Foundry attempts to simulate the terminal's display area, input type, and so on as closely as possible. Refer to Chapter 6, "Qlarity Foundry Preferences" for information.

Click to identify the version of Qlarity Foundry you are running.

3.6 Miscellaneous Bar

The miscellaneous bar next to the toolbar contains the Layout Scale and Outline All Objects functions in Layout View, an error indicator in Code View, and the Layout Scale in Simulation mode. Select **Misc. Bar** from the View menu to show or hide the miscellaneous bar functions.

Layout View Misc. Bar

To change the scale of the work area, click the drop-down arrow and select the scale from the drop-down list. This function is also available in Simulation mode.

Select this option to draw a line around all defined objects. This is useful when you want to know the exact border location of each object or when one or more objects were accidently moved out of the work area.

Code View Misc. Bar

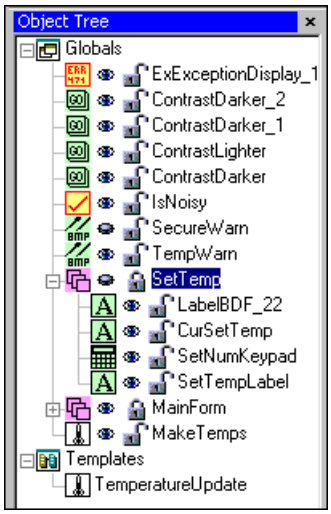
This box indicates if the selected line of code contains an error.

3.7 Object Tree

The Object Tree contains two branches:

- Globals
- Templates

The Globals branch is always visible; the Templates branch is only visible in Code View. The illustration below shows both branches of the Object Tree.



Each object instance listed in the Object Tree has an icon that shows the type of object (BDF label, button 3D, event timer, form, line, etc.). The objects are listed in Z-order (see Appendix A, "Glossary of Software Terms"), but you can move an object up or down the list using drag and drop.

The Object Tree also contains icons that can be used to show/hide or lock/unlock an object instance, as follows:

Click to show (eye open) or hide (eye closed) the object instance in the work area. This is useful if you have objects "stacked." You can hide an object if you want to see an object that is behind it.

Click to lock (lock closed) or unlock (lock open) an object instance. When an object is locked, it cannot be changed or moved with the mouse.

3.7.1 Globals

In Layout View (default view), you see only the Globals branch of the Object Tree. Under the Globals branch, all object instances defined for the workspace are listed.

To add a new object instance to a workspace, you can right-click "Globals" and select **New Object Instance** from the shortcut menu, or you can use the icons on the Object Palette.

If any global code has been written for the workspace, you can click Globals and see the object properties associated with the global code. Global properties apply to the entire workspace.

3.7.2 Object Templates

A Qlarity program consists primarily of objects. An object template contains the programming code that defines the object and how it behaves. Once object templates are created, any number of object "instances" based on the template can be added to a workspace. Each object instance in a workspace must be associated with a template.

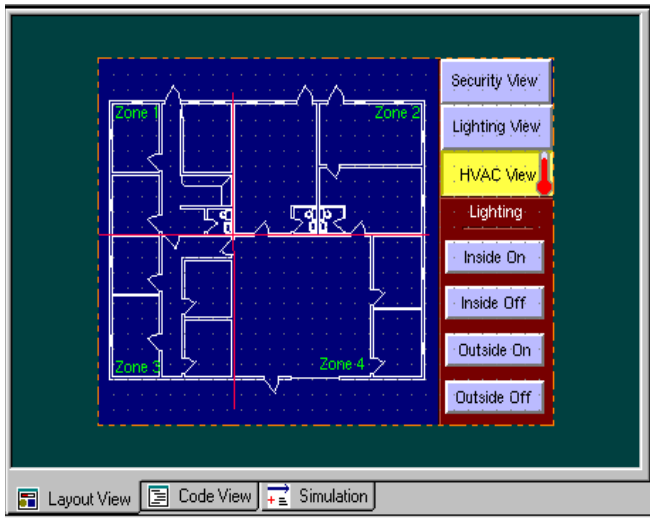
You will create object templates only if you are an advanced user. Most users create object instances from predefined templates in libraries.

NOTE: library object templates
Object templates in libraries are not listed in the Object Tree under the Templates heading. Only object templates created in the workspace are listed under the Templates heading.

3.8 Layout and Code Views

Layout View is the default view mode. You will generally use only Layout View unless you are customizing objects or creating your own objects using the Qlarity programming language.

An example of the work area of a workspace in Layout View is shown below.



Layout View closely simulates how the workspace will appear after it has been compiled and downloaded to the OptoTerminal.

If you understand the Qlarity programming language, you may use Code View to work with an object's programming

code. Select an object instance and click the Code View tab to display the property initializations for the object instance. Object instance code may also contain method overrides, which change the behavior of an object instance from that specified in the object template.

The following example shows the code for a text object in the sample workspace (shown in the Layout View illustration above).

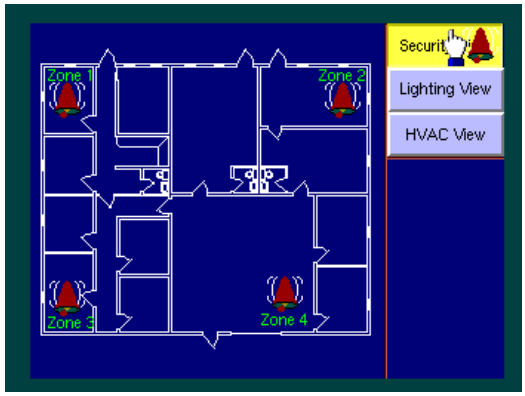
```
init parent := SetTemp
init xpos := 42
init ypos := 210
init width := 236
init height := 20
init caption := "Currently set to"
init font := helv14
init transparent := true

init fgcolor := rgb_white
```

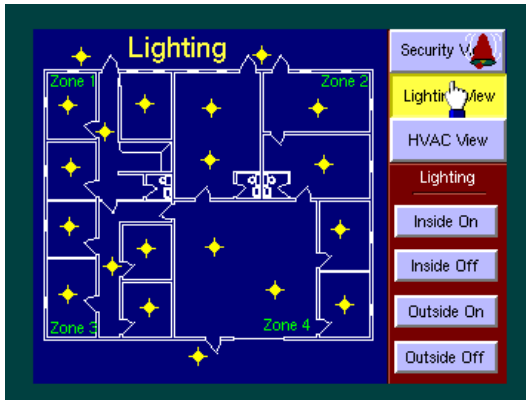
The following example shows the first portion of the programming code for the "TemperatureUpdate" template.

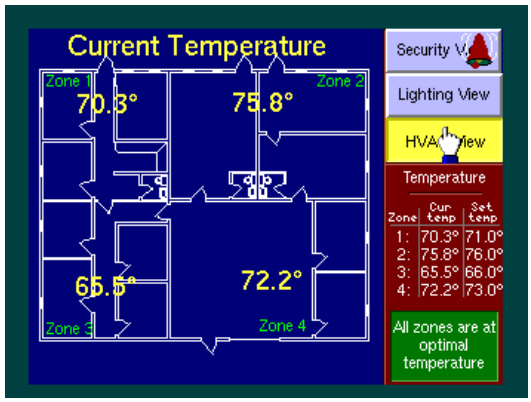
```
dim enabled as boolean
init enabled := true
func enabled(newval as boolean)
    enabled := newval
    enable(me,enabled)
endfunc

dim parent as objref
func parent(newval as objref)
    parent := newval
    attach(me,parent)
endfunc

'current temperature values
dim ctemp[4] as float
init ctemp := [71, 71, 71, 71]
func ctemp(newval[] as float)
    ctemp := newval
    updatecurtemp()
endfunc
func ctemp[](newval as float, index as integer)
    ctemp[index] := newval
```

3.9 Simulation

Simulation mode is used to simulate (with some limitations) the operation of a user application after it is downloaded to the terminal. Both touch screen and keyboard actions are simulated. On a simulated touch screen, for example, you can activate a button by clicking it with the mouse. When you click it, the programmed action occurs. In the following illustrations, the mouse pointer (a hand) is positioned over the activated button.



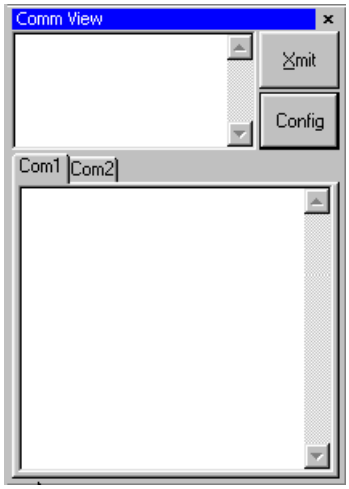




NOTE: required settings for simulation
To effectively simulate the OptoTerminal, you need to define the terminal's hardware in Qlarity Foundry, including whether you use a touch screen or keyboard. Use the Settings option to define the terminal's hardware and to set the simulation's volume and display functions. Refer to Chapter 6, "Qlarity Foundry Preferences" for information.

3.9.1 Serial I/O Support

Simulation mode partially supports serial I/O using the Comm View window (in the bottom-left corner of the display).



Serial data programmed to be transmitted through the terminal's Com1 or Com2 port appears in the lower portion of the Comm View window. You can also enter data in the upper portion of the window and click **[Xmit]** to simulate data being sent to the terminal.

Click **[Config]** to access the "Communications Window" settings (see section 6.5.2 for information).

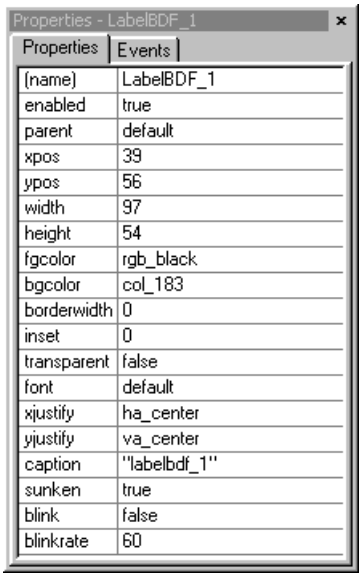
3.9.2 Simulation Limitations

The Simulation mode has the following limitations:

- When using a keyboard:
  - the repeat delay/rate is determined by your global setting in Windows. This will only match your terminal's settings if you set it to match on your PC.
  - PrintScreen and Pause/Break keys cannot be properly simulated.
- SystemSettings() (and related API functions) may not be fully simulated.
- The flash file system is simulated but not persistent (i.e., it is erased when you leave Simulation mode).
- The flash memory user data area is not saved when you close Qlarity Foundry.

3.10 Object Properties

Object Properties displays the properties of the selected object instance.



Click an object in the work area, or its object name in the Object Tree, to display its object properties. If Object Properties is not displayed, pull down the View menu, and click **Object Properties**.

Object properties are the data in an object instance. Examples of properties include:

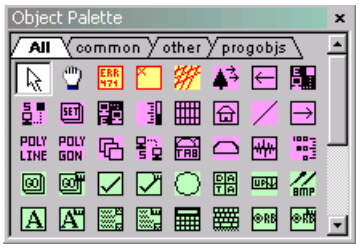
- Instance name (required)
- Enabled/disabled switch
- Parent (object to which the instance is attached, often a form)
- xy coordinates (x and y positions on the terminal display)
- Color (including color of background, text, border, etc.; selected from drop-down lists or the Select Color palette)
- Font (selected from a drop-down list)
- Bitmap image (selected from a drop-down list)
- Transparent (true/false)

You can edit property values by clicking the property you want to edit. Many properties have a drop-down list from which to choose a valid entry. You may also type in a value.

You can close Object Properties by clicking the Close button . Normally, however, you will want to keep Object Properties open.

3.11 Object Palette

The Object Palette contains a tab for each library in the workspace, as well as an "All" tab and an "Other" tab. Each tab has icons for all of the object templates in the library.



Click a tab to select an object from the specified library.

The "All" tab contains icons for all objects available to the workspace (from all of the libraries in the workspace). The background color of each icon identifies the library in which the object can be found. The "Other" tab contains objects from various small libraries.

For more information on object libraries, refer to section 5.3, "Edit Libraries."

Move the mouse pointer over an icon to display the object's name and description.

Click to place the mouse pointer in "select" mode. In this mode, click an existing object to select it. The pointer remains in select mode until you click a different icon.

Click to place the mouse pointer in "scroll" mode. If the layout scale is larger than the work area, use the scroll mode to view the parts of the workspace that are not visible. Click , then click anywhere in the work area and hold the mouse button as you "drag" in the direction you want to view. When the section you want to view is visible, release the mouse button. The pointer remains in scroll mode until you click a different icon or exit the workspace.

You can close the Object Palette by clicking the Close button , or you can show or hide it by clicking **Object Palette** on the View menu.

3.11.1 Add a New Object Instance

Do the following to add a new object instance from the palette.

1. Click the icon of the object instance you want to add.

2. Move the mouse pointer to the work area. The pointer changes to a cross hair.

3. Click and hold the mouse button, and drag the mouse to draw a rectangle in the work area. Then release the mouse button.

   Some objects have a default starting size and shape, so the size of the rectangle doesn't matter, only its position in the work area. The size of other objects, such as a line or rectangle, is initially determined by the size of the rectangle you draw.

NOTE: add multiple instances of the same object type
To add more than one instance of the same type of object, press and hold the <**Shift**> key when you click the object icon. The object type remains selected, and you can just click and drag in the workspace to add objects until you release the <**Shift**> key.

4. After you add an object, it remains selected and its object properties are displayed. You can then edit the properties as required, including assigning the object a unique name.

To resize an object, do one of the following.

- Use the mouse. If the object has "resize grips" (sizing handles), you can click and drag any handle to resize the object.

- Use Object Properties. Change the height and/or width integer in Object Properties. Select from the drop-down list, or type in the new value.

To move an object, do one of the following.

- Click in the center of the object, hold down the mouse button, and drag the object to a different position. You can hold down the <**Shift**> key while moving an object to limit its movement to horizontal, vertical, or a 45° angle.

- Change the x and/or y position integer in Object Properties. Select from the drop-down list, or type in the new value.

3.12 Move and Resize Windows

The following windows can be moved, resized, or undocked (separated) from the main window:

- Object Tree
- Object Properties (Layout View)
- Object Palette (Layout View)
- Messages and Errors
- Miscellaneous bar
- Communications window (Simulation mode)
- Keyboard (Simulation mode)
- Key Pad (Simulation mode)

Each window has its own title bar at the top (the Miscellaneous bar has a title bar on the left side under the Close button). To move a window, click the title bar, hold down the mouse button, and drag the window to another location, either in the main window or outside the main window. As you drag it, you will see an outline that indicates the position and size it will be. When it is in the position you want, release the mouse button. If you release the mouse button close to a "docked" position in the main window, the window snaps into place.

NOTE: floating windows
If you do not want the window to "snap" into a docked position, hold down the **<Ctrl>** key as you drag it. It then becomes a floating window that can be positioned anywhere on the screen.

To resize a window, move the mouse pointer over any side or corner until the pointer changes to directional arrows. Click and hold the mouse button, and drag the side or corner of the window until it is the size you want, then release the mouse button.

NOTE: resize the main window
You can resize the main window using the same method.

To close a window, click . To open a closed window, select its name from the View menu.

3.13 Where to Go From Here

When you have finished reviewing Getting Started, refer to Chapter 8, "Basic Design" to learn about the Qlarity design process, and specifically to section 8.4, "Event Builder" for information on Qlarity Foundry's user-friendly design tool for creating user applications. Refer to chapters 3 through 7, as required, to answer specific questions about using Qlarity Foundry.

CHAPTER 4

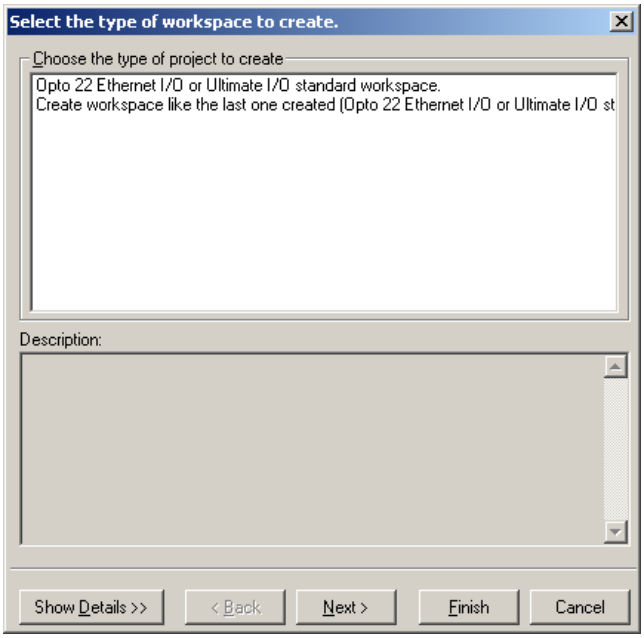
WORKSPACES

4.1 Start a New Workspace

When you first start Qlarity Foundry, the "Welcome to Qlarity Foundry" dialog box is displayed. To start a new workspace from this dialog box, click **[Start New]**.

To start a new workspace after Qlarity Foundry is loaded, click on the toolbar, or select **New Workspace** from the File menu.

The following dialog box is displayed.


Select the type of workspace to create.
Choose the type of project to create
Opto 22 Ethernet I/O or Ultimate I/O standard workspace.
Create workspace like the last one created (Opto 22 Ethernet I/O or Ultimate I/O st
Description:
Show Details >>
< Back
Next >
Finish
Cancel

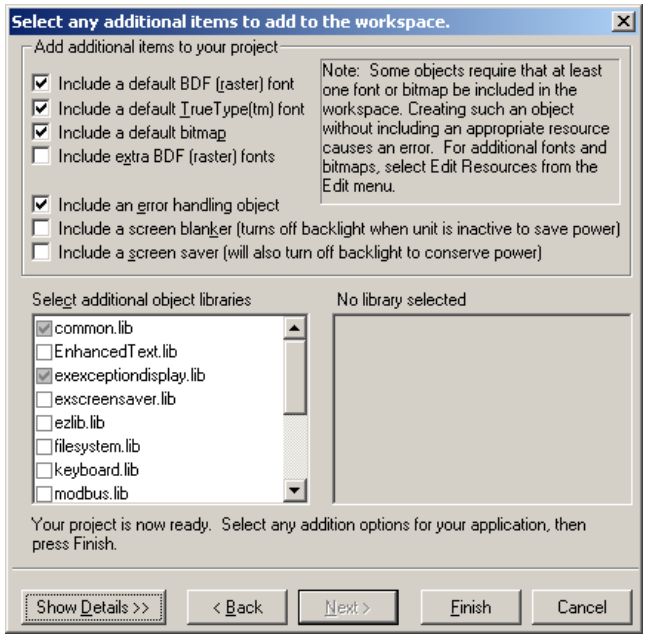
The upper box lists several types of workspaces you can create. The difference between workspace types is determined by the library or libraries included with the selection. New options are being added regularly, but at the time of this printing, the following workspace types were supported:

Opto 22 Ethernet I/O or Ultimate I/O standard workspace
This option includes the Common library and the Programmable Objects library. The Common library contains drawing, navigation, and communication objects required in all workspaces. The Programmable Objects library contains objects set up for intermediate level programming. For more information on the Programmable Objects library, refer to Chapter 9, "Intermediate Design."

Create workspace like the last one created
This option appears if you have previously created a workspace. Select this option to create a workspace containing the same libraries and resources that were in the last workspace created.

Click the workspace type that you want to use, and click **[Finish]** or **[Next]** (depending on the workspace option you selected).

If you click **[Next]**, the following dialog box is displayed.


Select any additional items to add to the workspace.
Add additional items to your project
Include a default BDF (raster) font
Include a default TrueType(tm) font
Include a default bitmap
Include extra BDF (raster) fonts
Note: Some objects require that at least one font or bitmap be included in the workspace. Creating such an object without including an appropriate resource causes an error. For additional fonts and bitmaps, select Edit Resources from the Edit menu.
Include an error handling object
Include a screen blanker (turns off backlight when unit is inactive to save power)
Include a screen saver (will also turn off backlight to conserve power)
Select additional object libraries
No library selected
common.lib
EnhancedText.lib
exexceptiondisplay.lib
exscreensaver.lib
ezlib.lib
filesystem.lib
keyboard.lib
modbus.lib
Your project is now ready. Select any addition options for your application, then press Finish.
Show Details >>
< Back
Next >
Finish
Cancel

This dialog box allows you to add or remove resources.

Include a default font and bitmap
Before some objects can be added to a workspace, at least one font or bitmap image must already exist in the workspace (if not, an error occurs). For this reason, it is recommended that you include the default fonts and bitmaps.

Include extra BDF (raster) fonts
To include additional fonts in your workspace, select this option.

Include an error handling object
By default, an error handling object (ExceptionDisplay) is added to your workspace that identifies and displays descriptions of any errors in your code. It is recommended that you include this object in all of your workspaces.

Include a screen blanker
By default, a ScreenBlanker object is added to your workspace that blanks the terminal's display after a period of time to save power and reduce wear on the backlight tube. By default, the display blanks after fifteen minutes, but you can change the default by selecting **ScreenBlanker** in the Object Tree and changing the value in Object Properties. If you don't want the display to blank, don't select this option.

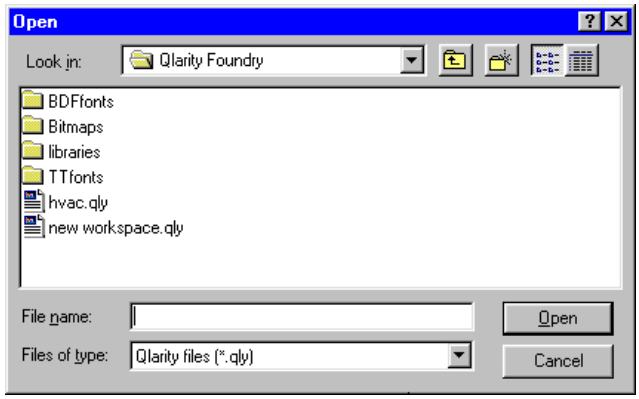
Include a screen saver
You can use a ScreenSaver object in the place of a ScreenBlanker object by selecting this option. Instead of blanking the terminal display, ScreenSaver displays a pattern of dancing lines for a period of time before it blanks the display. By default, the display switches to dancing lines after fifteen minutes and to a blank display after thirty minutes. You can change the default by selecting **ScreenSaver** in the Object Tree and changing the value in Object Properties.

When you are done, click **[Finish]** to proceed. The default workspace name is "New Workspace" until you save it with a new file name. The Object Palette includes tools to create all of the objects contained in the supported libraries.

NOTE: editing resources
You can add or remove resources and libraries after the workspace is created using the options on the Edit menu. Refer to section 5.2, "Edit Resources" and section 5.3, "Edit Libraries."

4.2 Open a Workspace

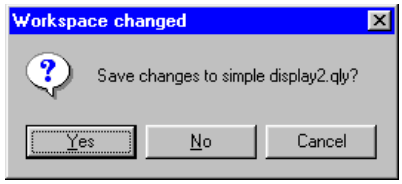
To open a workspace, click on the toolbar, or select **Open Workspace** from the File menu. The following dialog box is displayed.



Workspace files have a *.qly* extension. All workspace files in the default folder are listed (change folders if necessary). Click a file to select it, and its name is displayed in the "File name" field. Click **[Open]** to open the workspace in the main window. You can also double-click a file name to open it.

4.3 Close a Workspace

Select **Close Workspace** from the File menu to close the current workspace and start a new workspace. The following prompt appears.



Click **[Yes]** if you want to save the workspace that was loaded, or click **[No]** if you do not want to save it. A new, unnamed project is started using the default resources from the previous workspace. At this point, you can start a new project or open a different project using the **Open Workspace** option.

4.4 Save a Workspace

The first time you save a new workspace, the Save As dialog box appears so that you can enter a name for the workspace file (see section 4.4.2). Then, when you save the file, it is saved to the file name you entered.

If you want to make a copy of the workspace or change the file name, use "Save Workspace As."

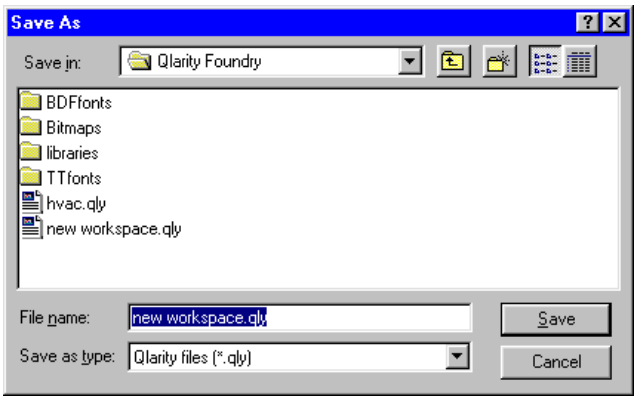
If you want to transfer a workspace to another computer, use the "Collect for Output" function (see section 4.4.3 below).

4.4.1 Save Workspace

To save a workspace, click on the toolbar or select **Save Workspace** from the File menu. If you have already named the workspace, the file is saved. If it is a new workspace and you have not yet named it, the Save As dialog box appears (see section 4.4.2).

4.4.2 Save Workspace As

To save a workspace to a different name, select **Save Workspace As** from the File menu. The following dialog box is displayed.



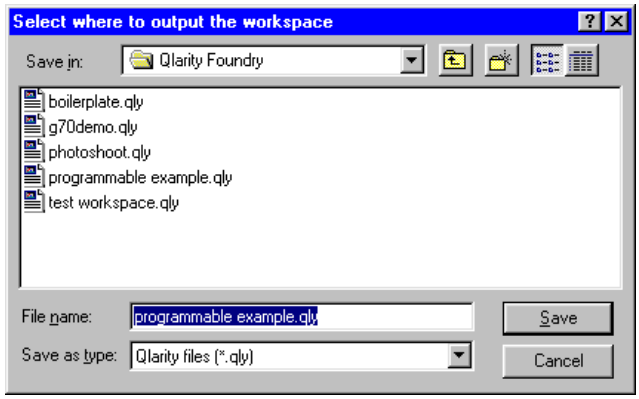
In the "File name" field, type the name that you want to give the workspace (you do not have to type the extension). Click **[Save]** to save the workspace.

The original file is retained. The new file is a copy of the original file. If you are renaming the file, you will need to delete the original file.

4.4.3 Collect for Output

Use this option to collect all of the resources used in a workspace into one file. You can then move the file to another computer, and all resources will be available to open the workspace.

To transfer a workspace to another computer, select **Collect for Output** from the File menu. The following dialog box is displayed.



In the "File name" field, type the name that you want to give the output file (you do not have to type the extension). Click **[Save]** to save the file.

4.5 Compile a Workspace

Click on the toolbar, or select **Compile Workspace** from the File menu to compile the current workspace into the format required for a user application. Compiling is also required to update modifications made in Code View before they can be seen in Layout View. However, if you make changes in Code View and then switch to Layout View, the program compiles automatically.

If the compile is successful, the program displays the application in Layout View. If any errors occur during the compile, the program switches to Code View and the error messages are displayed in the Compile dialog box.

4.6 Generate a BFF File

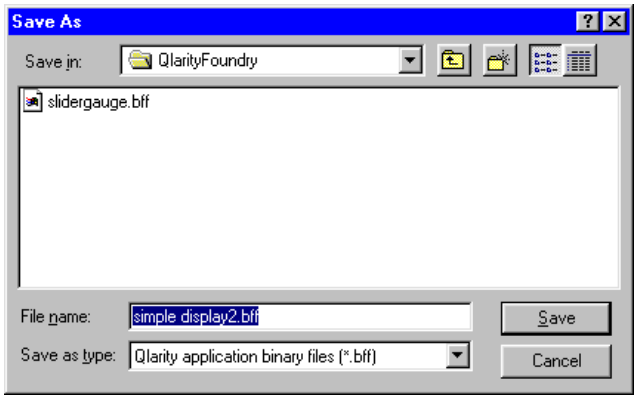
Use this function to save a Qlarity workspace file to binary file format (BFF), which is the application format required by the terminal.

When you use the "Download Application" option to download the current workspace to the terminal, Qlarity Foundry automatically compiles the workspace into a BFF file; however, the BFF file is not saved.

To save a workspace's BFF file so that you can download it at any time without loading the workspace in Qlarity Foundry, do the following.

1. Load the workspace that you want to save to a BFF file.

2. Select **Generate [*filename*].bff** from the File menu (where **[*filename*]** is the name of the current workspace). The following dialog box is displayed.



3. In the "File name" field, type a name for the BFF file, or accept the default file name (you do not have to type the extension). The workspace file name with a *.bff* extension in place of the *.qly* extension is used as the default.

4. Click **[Save]** to generate the BFF file.

You can download the BFF file using the **[Download Other BFF]** option in the Download Application function (File menu). Refer to section 7.3, "Download a BFF File" for information.

CHAPTER 5

TEMPLATES, RESOURCES, AND LIBRARIES

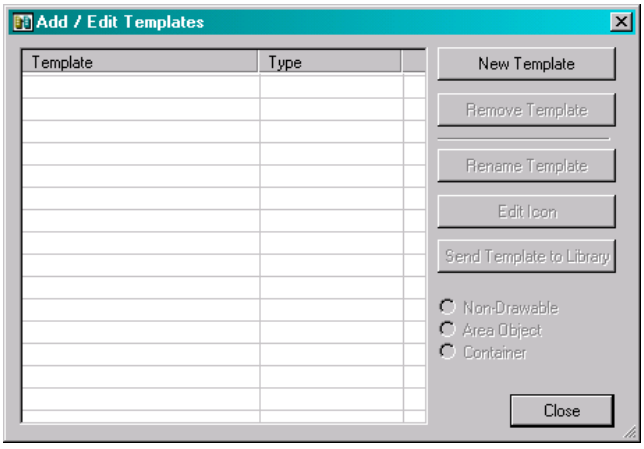
5.1 Add/Edit Templates

Creating and designing object templates is an advanced function, which is covered in depth in Chapter 10, "Advanced Design." This section provides an overview of the Add/Edit Templates function.

Use Add/Edit Templates to create a new object template. The new template may be blank, it may contain only boilerplate code, or it may be based on a library object template. New object templates are added to the Template branch of the Object Tree. After adding a template, you use Code View to add or edit your custom programming code.

You can also use the Add/Edit Templates function to change the name of an existing template.

Click on the toolbar, or select **Add/Edit Templates** from the Edit menu, and the Add/Edit Templates dialog box is displayed. (You can also right-click anywhere in the Templates branch of the Object Tree, and click **Add/Edit Templates** on the shortcut menu).



5.1.1 Add a New Object Template

In the Add/Edit Templates dialog box, do the following to create a new object template.

1. Click **[New Template]** and the following dialog box is displayed.

2. Select one of the following options to start the new template:

   Create a template ready to operate in Qlarity Foundry
   This option creates a new object template with sufficient programming code to function as an object and display on the screen. Additional programming is required to define the object's function, appearance and so on.

   Create a completely blank template
   This option creates a blank, unprogrammed object template.

   Create a template based on a pre-existing library object
   This option creates a new object template using the programming code from a library object you specify in the "Based on" box.

3. In the "Template Name" box, type a name for the new template. Each template in the workspace must have a unique name. A template name has no size limitation but must start with a letter. A name cannot contain spaces but may use the underline character (_). The percent (%), pound (#) and dollar sign ($) symbols can be used at the end of the name.

4. The last box is labeled either "Object Type" or "Based on" depending on the option you selected.

   In the "Object Type" box, select the type of object you are creating from the drop-down list, as follows:

Container	An object that contains other objects (e.g., a form). It can be defined as part or all of the terminal screen.
Area object	A display element (e.g., text object, bitmap object, line object, etc.)
Non-drawable object	An object that serves a function not related to the display (e.g., key definition, communications, etc.).

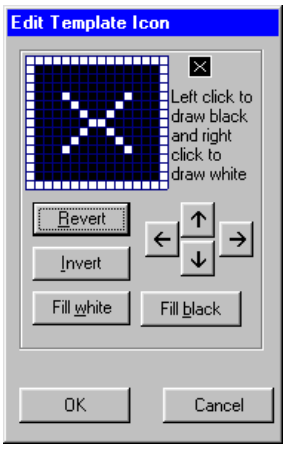
   For more information on object types, refer to Chapter 8, "Basic Design" and to Appendix A, "Glossary of Software Terms."

   In the "Based on" box, select the library object on which the new object template is to be based. The programming code from the library object will be copied to the new object template.

5. Click **[OK]** to create the new template.

5.1.2 Edit Icon

Each template is identified in the Object Tree by an icon. To customize the icon for a template, click the template name in the Add/Edit Templates dialog box, then click **[Edit Icon]**. The following dialog box is displayed.



The icon form is graphically represented as a grid. Each square in the grid is an individual pixel. On the grid, draw the icon for the template using the following tools.

- Click in a pixel to fill it with black. Right-click in a pixel to fill it with white. Drag to fill several consecutive pixels.
- Click Fill white to fill all pixels in the grid with white.
- Click Fill black to fill all pixels in the grid with black.
- Click ↑ ← → ↓ to move or offset the graphic in the grid. For example, you can click Fill white, then click ← twice and click → once to add side borders, and click ↑ twice and ↓ once to add top and bottom borders.
- Click Revert to return the graphic to its original design.
- Click Invert to reverse the black and white pixels.

5.1.3 Rename Template

In the Add/Edit Templates dialog box, do the following to change an object template's name.

1. Click the template name that you want to edit. It is highlighted in blue.
2. Click **[Rename Template]**.
3. Type the new name over the old name.
4. Click **[Close]** to save the change and exit.

5.1.4 Remove Template

In the Add/Edit Templates dialog box, do the following to remove an object template from the workspace.

1. Click the template that you want to remove. It is highlighted in blue.
2. Click **[Remove Template]**.
3. A prompt is displayed to confirm that you want to remove the template. Select **[Yes]** to remove the object template from the workspace.

5.1.5 Send Template to Library

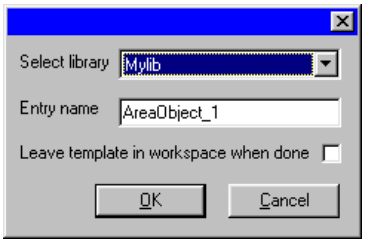
After you have created and customized an object template, you can add it to a library. By placing your object templates in a library, they can easily be distributed to other locations or computers by copying the library file (e.g, *libraryname.lib*).

NOTE: do not add to Opto 22 libraries
You should not save your new object templates to Opto 22 libraries. Opto 22 libraries are regularly updated, and the new libraries replace the existing ones, so your templates would be overwritten. Instead, you should create a new library in which to store your own object design work. Refer to section 5.3, "Edit Libraries" for more information.

In the Add/Edit Templates dialog box, do the following to add an object template to a library.

1. Click the template that you want to add to a library. It is highlighted in blue.

2. Click **[Send Template to Library]**. The following dialog box is displayed.



3. In the "Select Library" box, select the library to which you want the object template copied.

4. The "Entry Name" box displays the name of the object template. To assign it a different name within the library, type a new name.

5. If you want to leave the template in the current workspace after you copy it to the library, select **Leave template in workspace when done**.

6. Click **[OK]** to place the object template in the selected library.

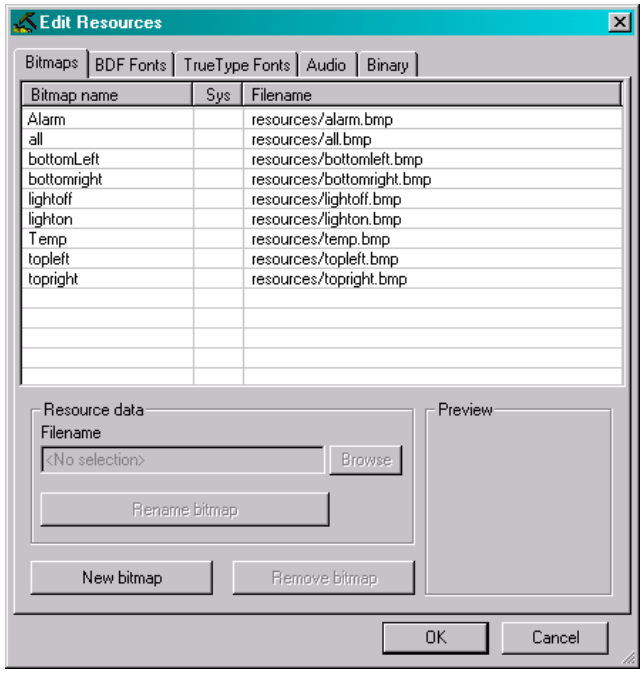
5.2 Edit Resources

Resources include all bitmap images, fonts, audio files, and binary data files that are available in a workspace and, eventually, in the user application. You must add resources to a workspace before you can use them in object instances.

Use the Edit Resources option to add resource files to your workspace or to change or remove the resources available. Because resources become part of the user application when it is compiled, you should delete any unnecessary resources.

NOTE: default bitmaps and fonts
When you create a new workspace, you have the option to add default bitmap images and fonts to your workspace. Some objects may require at least one bitmap, BDF font, or TT font resource before the object will function properly. When you have finished creating a workspace, however, you may want to remove any unused default bitmaps and fonts before you download the user application to the terminal to conserve the terminal's flash and RAM memory.

Click on the toolbar or select **Edit Resources** from the Edit menu, and the Edit Resources dialog box is displayed.



All resources used in a workspace are managed from this dialog box.

Click the tab for the resource type you want to manage, then follow the procedures in the following sections as applicable to add a new resource, rename an existing resource, change the file for an existing resource name, or remove a resource from the workspace. Additional information on each resource type is also provided in the following sections.

After you have finished adding, editing, or removing resources, click **[OK]** to save your changes and exit the Edit Resources dialog box. The workspace is compiled. Resources that you added are available for use in the objects, and resources that you removed are no longer available.

Save the workspace to save the new information to disk.

NOTE: button labels
The labels on the buttons used to edit resources change to match the resource tab you selected. For example, you click **[New bitmap]** to add a bitmap image, **[New BDF font]** to add a BDF font, and **[New audio resource]** to add an audio file. The label on the "Rename" button changes to the name of the resource you selected to rename.

5.2.1 Add a Resource

In the Edit Resources dialog box, do the following to add a new resource to the workspace.

1. Click **[New *resource type*]** (e.g., **[New BDF Font]**) and an Open dialog box appears. Resource files in the default folder are listed (change folders if necessary).

2. Select the resource file or files that you want to add.

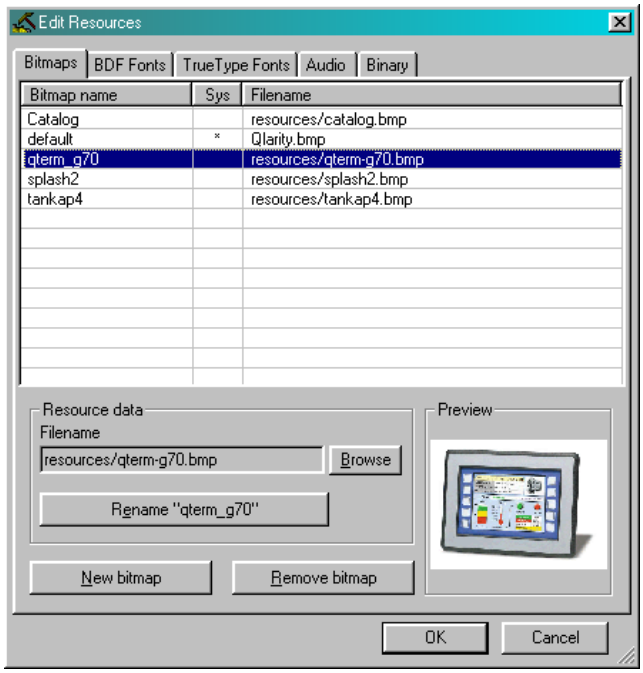
NOTE: select multiple files
To select multiple files that appear consecutively in a list, press and hold **<Shift>**, then click the first and last file in the group you want to select. To select multiple non-consecutive files, press and hold **<Ctrl>**, then click each file that you want to select.

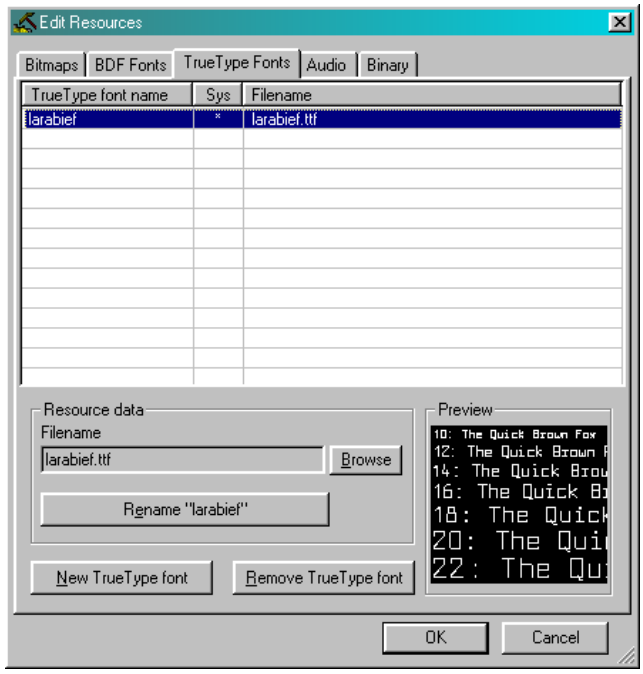
3. Click **[Open]**. The resource is added to the list of resources.

When you add a resource, it is given the name of the file by default. The resource name is the name that appears in Object Properties for you to select the resource for an object instance. For information on changing the name, refer to section 5.2.3, below.

5.2.2 Preview Resources

When you select a resource in the list, a preview of the resource is displayed in the "Preview" box (as shown below) if appropriate. If you selected an audio resource, click the button in the "Preview" box to listen to the sound.

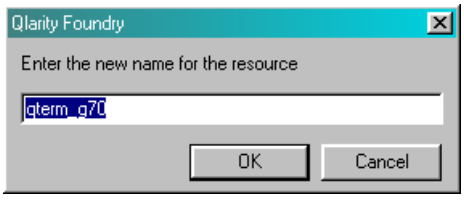




5.2.3 Rename a Resource

In the Edit Resources dialog box, do the following to change the name of a resource. This changes the name that appears in Object Properties for you to select the resource for an object instance; it does not change the file name of the resource.

1. From the list of resources, select the resource that you want to rename, and click **[Rename "*resource name*"]** (e.g., **[Rename "OptoTerminal_g70"]**). A dialog box is displayed with the current name of the resource selected in the text box.



2. Type over the name with the new name, or edit the name as desired.

3. Click **[OK]** to rename the resource and close the dialog box. The new name appears in the "name" column of the resource list.

5.2.4 Change a Resource File

In the Edit Resources dialog box, do the following to change the file assigned to an existing resource name.

1. From the list of resources, select the resource for which you want to change the file.

2. Click **[Browse]** and an Open dialog box appears.

3. Select the file that you want to assign to the resource name.

4. Click **[Open]** to accept the file and close the dialog box, and the file is changed.

5.2.5 Remove a Resource

In the Edit Resources dialog box, do the following to remove a resource from the workspace.

1. From the list of resources, select the resource or resources that you want to remove. You can select multiple resources as described in the note above.

2. Click **[Remove *resource type*]** (e.g., **[Remove BDF Font]**). The selected resource(s) is removed from the workspace.

5.2.6 Bitmaps

Qlarity Foundry supports bitmap images in the standard *.bmp* format. Use the following guidelines when creating bitmap images to be displayed on the OptoTerminal:

Bitmap Size:
The size of a bitmap image is normally shown as the number of pixels in width by the number of pixels in height (e.g., 25 x 10). The OptoTerminal-G70 display is 320 pixels in width by 240 pixels in height (320 x 240), and the OptoTerminal-G75 is 640 pixels in width by 480 pixels in height (640 x 480). Therefore, you can easily determine what size to make your bitmap images so they will be in proportion to the size of your terminal. Refer to your terminal's specifications for its exact size.

Transparent Background:
You can specify that a color in a bitmap image become transparent in the workspace. If you want the background of a bitmap image to be transparent in the workspace, make the background a color that is not used in the bitmap, then select the background color as the transparent color in the bitmap object properties in Qlarity Foundry. Bright magenta and bright green are often used as transparent background colors.

5.2.7 Fonts

OptoTerminals support BDF (raster) and TrueType fonts. Use the following guidelines to help you determine which fonts to use in a workspace:

Scalable and Non-scalable Fonts:
BDF (raster) fonts are non-scalable, which means that you need to load a separate font file for each point size (e.g., Arial 8, Arial 14, Arial 20, etc.). TrueType (TT) fonts are scalable, which means that a single font can typically be scaled to several different point sizes.

Rotating a Font:
A TT font can be rotated to any angle. A BDF font can only be displayed at the angle at which it was created.

Availability:
BDF fonts and TrueType fonts are readily available from several sources, including many public domain Internet sources. TrueType fonts can be used on any Windows computer.

Terminal Memory Use:
A BDF font file is smaller than a TT font file. If you are using only one or two sizes of a specific font (Arial, Times Roman, etc.), you can save terminal memory using BDF fonts. If you are using several point sizes for a specific font, the TT font will likely use less memory than three or more BDF fonts.

Display Speed:
BDF fonts are displayed more quickly than TrueType fonts on the OptoTerminal. While the terminal renders TrueType fonts fast enough that the difference in drawing speed is rarely noticeable, you might consider using BDF fonts for display items that will change their text frequently (such as items that are frequently polled from a remote device).

Copyright:
Many fonts are copyrighted. Be sure to purchase or obtain permission to use a copyrighted font in your user applications.

5.2.8 Audio

Some terminals from QSI Corporation have an audio codec option that enables them to play waveform audio files. If your terminal supports an audio codec, you can add audio files in the *.wav* format to your workspace. You can play the audio files using the PlaySound API function.

5.2.9 Binary

Binary resources are files that you want to add to your user application unchanged. For example, you could use a binary resource in your application if you wanted the terminal to communicate with an embedded device (with no PC connection). You could, for example, add a configuration or firmware file to the application as a binary resource file, and then use the GetBinaryResource API function to download the file from the terminal to the embedded device when the application is running.

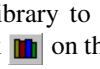
5.3 Edit Libraries

Object libraries are provided by Opto 22 (and other sources) to assist you in designing user applications. Each library contains several predefined Qlarity objects. This enables you to add object instances to your workspaces without the need to program new object templates.

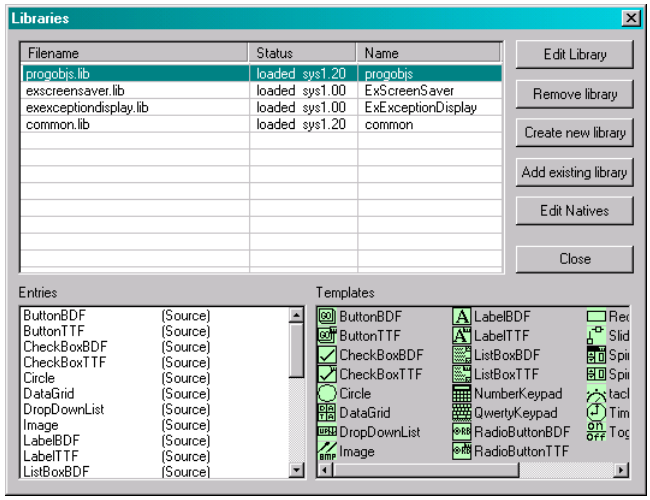
When you add a library to your workspace, the objects in the library are added to the Object Palette and to the drop-down list in the Instance Properties dialog box (right-click anywhere in the Object Tree, and click **New Object Instance**).

NOTE: default libraries
When you create a new workspace, you have the option to add default libraries to the workspace. You can remove a default library or add new libraries to the workspace using the Edit Libraries option.

NOTE: library object templates
Object templates in libraries are not listed under "Templates" in the Object Tree. Only object templates created in the workspace are listed under this heading. However, you can create a new object template based on an existing library object (see section 5.1, "Add/Edit Templates" for more information).

To add a library to or remove a library from your workspace, click  on the toolbar, or select **Edit Libraries** from the Edit menu.

A dialog box similar to the one below is displayed.



All libraries currently in the workspace are listed. Click a library to display its entries and object templates at the bottom of the dialog box.

5.3.1 Add Existing Library

Do the following to add a new library to the workspace.

1. Click **[Add Existing Library]**, and a dialog box appears that lists all the libraries included with Qlarity Foundry that have not been added to the workspace.

2. Select one or more libraries from the list.

   To select multiple libraries that appear consecutively in the list, press and hold **<Shift>**, then click the first and last library in the group you want to select. To select multiple non-consecutive libraries, press and hold **<Ctrl>**, then click each library that you want to select.

   The objects in the selected library or libraries are displayed at the bottom of the dialog box.

3. Click **[OK]** to close the dialog box, and the selected libraries are added to the workspace. The objects in the new libraries are added to the Object Palette.

4. To add a library that is not listed, click **[Add Other Library]** and an Open dialog box appears. Select the library or libraries that you want to add (change the folder if necessary), and click **[Open]**. The library is added to the workspace.

5. Click **[Close]** to save your changes and exit the dialog box. The workspace is compiled and the objects in the new library are available for use in the workspace.

6. Save the workspace to save the information to disk.

5.3.2 Remove Library

Do the following to remove a library from the current workspace.

1. All libraries that are loaded in the workspace are listed. Click the library that you want to remove. It is highlighted.

2. Click **[Remove Library]** and the library file is removed from the workspace.

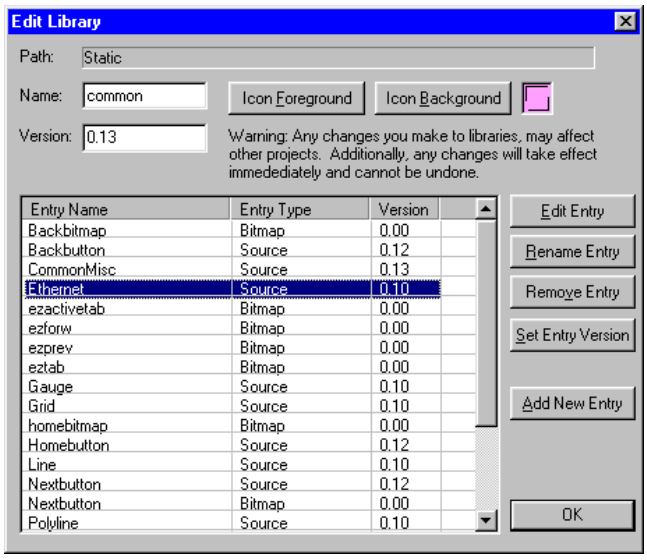
3. Click **[Close]** to save your changes and exit the dialog box. The workspace is compiled and the objects in the library are no longer available for use in the workspace.

5.3.3 Edit Library

Library editing functions require an understanding of Qlarity programming. Refer to Chapter 9, "Intermediate Design" and Chapter 10, "Advanced Design" for information on Qlarity programming.

NOTE: do not edit Opto 22 libraries
You should not edit Opto 22 libraries. Opto 22 libraries are regularly updated, and the new libraries replace the existing ones, so any changes are overwritten. You should edit only libraries that you created (see section 5.3.5, "Create a New Library"). You can modify an object template by importing it into your workspace (see section 5.1.1, "Add a New Object Template").

In the Libraries dialog box, click a library to select it, then click **[Edit Library]**, and the following dialog box is displayed.


Edit Library
Path: Static
Name: common
Icon Foreground
Icon Background
Version: 0.13
Warning: Any changes you make to libraries, may affect other projects. Additionally, any changes will take effect immedediately and cannot be undone.
Entry Name | Entry Type | Version
Backbitmap | Bitmap | 0.00
Backbutton | Source | 0.12
CommonMisc | Source | 0.13
Ethernet | Source | 0.10
ezactivetab | Bitmap | 0.00
ezforw | Bitmap | 0.00
ezprev | Bitmap | 0.00
eztab | Bitmap | 0.00
Gauge | Source | 0.10
Grid | Source | 0.10
homebitmap | Bitmap | 0.00
Homebutton | Source | 0.12
Line | Source | 0.10
Nextbutton | Source | 0.12
Nextbutton | Bitmap | 0.00
Polyline | Source | 0.10
Edit Entry
Rename Entry
Remove Entry
Set Entry Version
Add New Entry
OK

NOTE: use caution in editing libraries
Changes that you make to libraries take effect immediately and cannot be undone. Remember that a library may be used in more than one workspace, and changes made to the library affect every workspace in which it is loaded.

You can change the library name, version or icon colors, as well as change the entries in the library.

Name
To change the library's name, click in the "Name" text box, and type over the existing entry. The change is reflected in the "Filename" column in the Libraries dialog box.

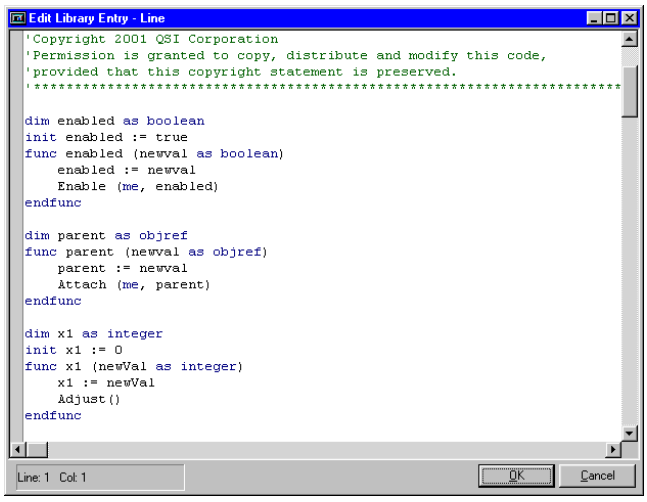
Version
To change the version number of the library, click in the "Version" text box, and type over the existing number. The change is reflected in the "Status" column in the Libraries dialog box.

Icon Foreground/Icon Background
On the Object Palette, icons for all objects in a library use a common color scheme. Use these options to specify or change the foreground and/or background color of the library's object icons. Click **[Icon Foreground]** or **[Icon Background]** and a color palette is displayed. A rectangle appears around the current foreground or background color. Click a different color square to change the color, then click **[OK]**.

5.3.3.1 Edit Entry

Click an object template entry in the "Entry Name" list to select it, then click **[Edit Entry]** to edit the programming code of the selected object. This function is only available for "source" type entries.

The object template is displayed in a text editor similar to Code View, as shown below.



Edit the code, and click **[OK]** when you are finished to save the changes.

5.3.3.2 Rename Entry

Click an entry in the list to select it, then click **[Rename Entry]** to change the name of the selected entry. A dialog box is displayed for you to type the new name. Click **[OK]** to apply the change and close the dialog box.

5.3.3.3 Remove Entry

Click an entry in the list to select it, then click **[Remove Entry]** to remove the selected entry from the library. A message is displayed warning you that this will permanently remove the entry from the library. Click **[Yes]** to remove the entry.

5.3.3.4 Set Entry Version

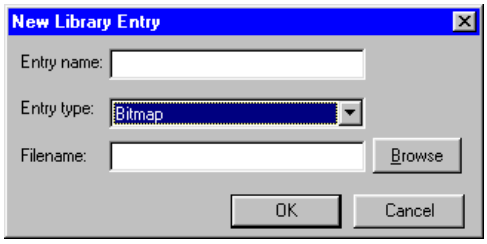
Click an entry in the list to select it, then click **[Set Entry Version]** to change the version number of the selected entry. A dialog box is displayed for you to type the new number. Click **[OK]** to apply the change and close the dialog box.

5.3.3.5 Add New Entry

A library contains two types of entries:

- Source objects (object templates)
- Embedded resources (e.g., bitmaps, BDF fonts, and TT fonts). Only resources that are used in an object template, (e.g., a custom button, tab, etc.) need to be embedded in the library. Libraries can also reference any resource that has been loaded into a workspace. However, by embedding the resource in the library, you make certain that it is available when an object instance is created.

To add a new entry to a library, click **[Add New Entry]**. The following dialog box is displayed.



Entry Name
Enter a name for the new entry. A name has no size limitation but must start with a letter or underline character (_). A name cannot contain spaces but may contain the underline

character. The percent (%), pound (#) and dollar sign ($) symbols can be used at the end of the name.

Entry Type
From the drop-down list, select the type of entry:

- Source (object template)
- Bitmap (resource)
- BDF font (resource)
- TT font (resource)

Filename
If the library entry is a resource (e.g., bitmap or font) enter the location of the resource file. Click **[Browse]** to find the file if necessary.

Click **[OK]** to continue. If the library entry is a "source" (object template), a message is displayed explaining that Qlarity Foundry will insert a blank entry in the library. To enter the programming code for the object template, select the new entry and click **Edit Entry**.

You can also add new object templates that you have already created with the "Send Template to Library" option in Add/Edit Templates (see section 5.1.5) for information.

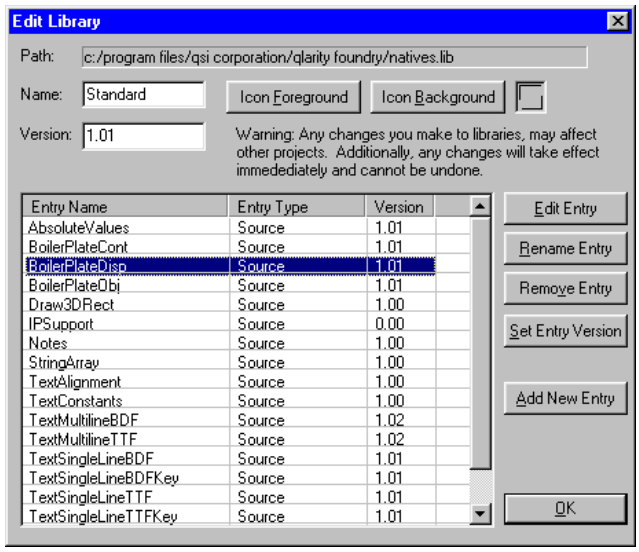
5.3.4 Edit Natives

In addition to Native API functions (as described in the *Programmer's Reference Manual*), the standard Natives library (Natives.lib) contains several utility functions written in the Qlarity language. To make these functions available, type the following in the Globals section of your workspace:

```
library standard source <entryname>
```

<*entryname*> is the name of the library that contains the additional API functions you want to use. To determine the entry in which a function resides, locate the function in the Object Documentation Help file. Each utility function in the library begins with the underscore (_) character.

Click the **[Edit Natives]** button to view the utility functions included in the Natives library.


Edit Library
Path: c:/program files/qsi corporation/qlarity foundry/natives.lib
Name: Standard
Icon Foreground
Icon Background
Version: 1.01
Warning: Any changes you make to libraries, may affect other projects. Additionally, any changes will take effect immedediately and cannot be undone.
Entry Name | Entry Type | Version
AbsoluteValues | Source | 1.01
BoilerPlateCont | Source | 1.01
BoilerPlateDisp | Source | 1.01
BoilerPlateObj | Source | 1.01
Draw3DRect | Source | 1.00
IPSupport | Source | 0.00
Notes | Source | 1.00
StringArray | Source | 1.00
TextAlignment | Source | 1.00
TextConstants | Source | 1.00
TextMultilineBDF | Source | 1.02
TextMultilineTTF | Source | 1.02
TextSingleLineBDF | Source | 1.01
TextSingleLineBDFKey | Source | 1.01
TextSingleLineTTF | Source | 1.01
TextSingleLineTTFKey | Source | 1.01
Edit Entry
Rename Entry
Remove Entry
Set Entry Version
Add New Entry
OK

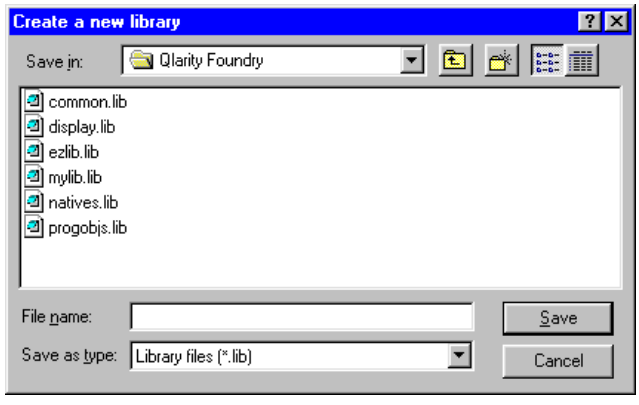
Do not modify these functions, as objects in various libraries also use them. In addition, any changes you make to Natives library functions are overwritten when you upgrade Qlarity Foundry.

Refer to section 5.3.3, "Edit Library" for information on editing functions.

5.3.5 Create a New Library

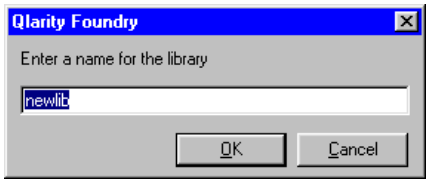
Do the following to create a new Qlarity Foundry library,

1. Click **[Create New Library]**. A dialog box similar to the one shown below is displayed.



2. Enter a file name for the new library (e.g., *mylib*) and click **[Save]**. The extension (*.lib*) is added automatically.

The following dialog box is displayed.



3. Enter a descriptive name for the library (e.g., "arch library" or "Johns library") and click **[OK]**.

The new library file is created and added to the Qlarity Foundry folder. Use the Edit Library function to add entries (object templates and resources) to the library (see section 5.3.3.5, "Add New Entry" for information).

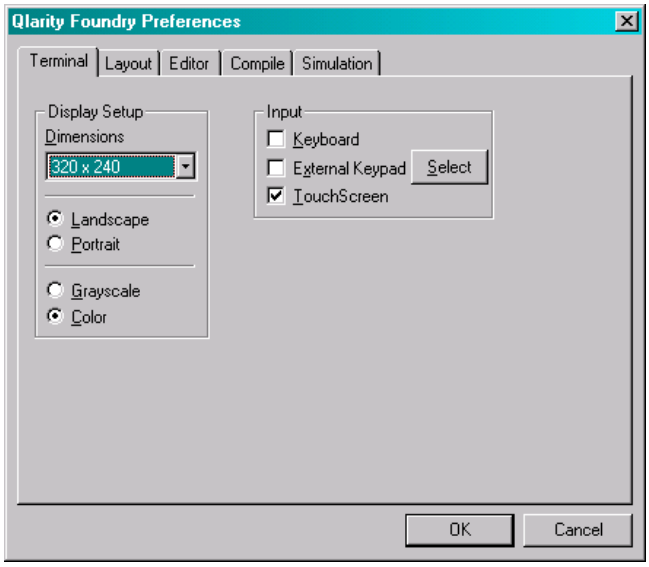
You can also add new object templates to a library using the "Send Template to Library" option in Add/Edit Templates (see section 5.1.5 for information).

CHAPTER 6

QLARITY FOUNDRY PREFERENCES

Use the Settings option to record the display configuration of your OptoTerminal, and to enter Qlarity Foundry preferences. Qlarity Foundry requires this information to simulate as accurately as possible the terminal display. The Settings option is also used to set up options for the code editor (Code View) and to enable or disable warnings and other defaults for compiling.

Click on the toolbar, or select **Settings** from the View menu, and the Qlarity Foundry Preferences dialog box is displayed.



Tabs are available to define your terminal's display, to define the layout of the Qlarity Foundry work area, to set up the code editor, and to set up compile defaults, as described in the following sections.

6.1 Terminal

Click the Terminal tab (shown above) to define the following options on your OptoTerminal's display.

NOTE: Simulation mode
You cannot modify the settings in this tab in Simulation mode.

6.1.1 Display Setup

Dimensions
From the drop-down list, select the display dimensions (in pixels) of your terminal model. The OptoTerminal-G70 has a 320x240 pixel display, and the OptoTerminal-G75 has a 640x480 pixel display. Refer to the specifications for your terminal for more information.

Landscape/Portrait
A terminal may be mounted in **Landscape** (longest dimension is horizontal) or **Portrait** (longest dimension is vertical) mode. Click the applicable mode for your terminal.

Grayscale/Color
Click the applicable color mode of your terminal, **Grayscale** or **Color**.

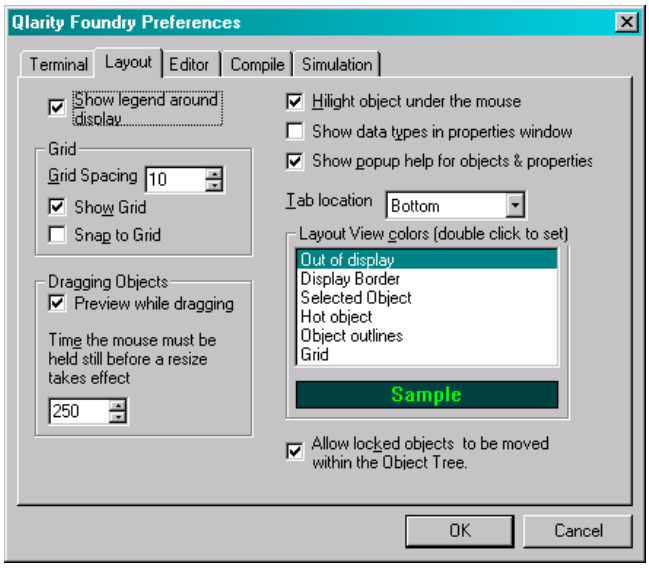
6.1.2 Input

Keyboard/External Keypad/TouchScreen
Click the input mode or modes that are supported by your terminal and that you plan to use.

If you select **[External Keypad]**, click **[Select]** to select an external keypad configuration, create a new configuration, or modify an existing one to match the actual keypad that you are using with the terminal.

6.2 Layout

Click the Layout tab to define the layout of the Qlarity Foundry work area to simulate your terminal display.



Show legend around display
With touch screen displays, you can add a border around the display for a stick-on, touch key legend. If you are using a touch key legend, select this option. If you are not using a touch key legend, leave the checkbox empty.

Grid Spacing
You can display a grid over the work area to help you more accurately place and align objects. Enter the space between grid lines in pixels (between 2 and 25).

Show Grid
Select this option to make the work area grid visible.

Snap to Grid
Select this option to force objects to "snap" to the nearest grid line when added or moved. This is useful for more accurate placement and alignment of objects.

Preview while dragging
(only available in Windows 2000 or Windows XP) This option controls the way objects appear while dragging them with the mouse to move them in Layout View. If selected, a preview of the object in the new location is displayed. If not selected, an outline of the object is displayed.

Time the mouse must be held...
This setting determines how long the redraw function is delayed when dragging an object. For most systems, the default should be optimum. If you need to increase or decrease the time interval, enter a value in milliseconds.

Highlight object under the mouse
If this option is enabled, each object's outline is highlighted as you move the mouse pointer over it. If disabled, an object's outline only appears when you click it.

Show data types in properties window
If this option is enabled, the data type of each property (integer, color, string, boolean, bdf font, etc.) is shown in Object Properties.

Show pop-up help for objects and properties
If this option is enabled, a Help pop-up appears when you move the mouse pointer over icons in the Object Palette and when you click on properties in Object Properties. The text for the pop-up Help is based on the Object Documentation file, which you can view by pressing **<F1>**.

Tab Location
From the drop-down list, select your preferred location in the window for the Layout View and Code View tabs.

Layout View Colors
The components that appear in the work area layout are listed. Click a component to display an example (in the "Sample" box) of the color in which it appears in the work area. Double-click an item to change the color. A color selector dialog box appears. Click a color in the basic colors section to select it. To create a custom color, drag the color selector in the rainbow palette to a color and then drag the slider up or down to adjust the saturation and lightness of the color; or enter the HSL or RGB of a color. Click **[OK]** to change the component to the selected color.

The work area components include the following:

Out of Display
The area that the legend covers (not shown if "Show Legend Around Display" is enabled).

Display Border
The line separating the border and the work area (not shown if "Show Legend Around Display" is enabled).

Selected Object
An object that you click. The selected color outlines the object to show that it is selected.

Hot Object
The object under the mouse pointer if "Highlight object under the mouse" is selected. The selected color outlines the object as the pointer moves over it.

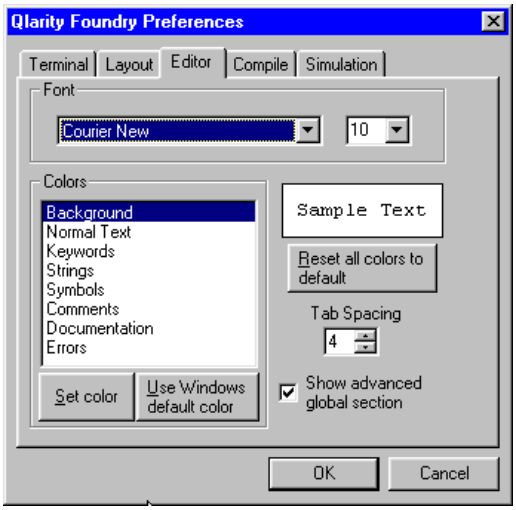
Object Outlines
The outlines around objects if "Outline all objects" on the toolbar is enabled.

Grid
Grid lines appear in the selected color if "Show Grid" is enabled.

Allow Locked Objects to be Moved within Object Tree
If an object is locked, it cannot be moved in the work area. If this function is enabled, however, a locked object can be moved in the Object Tree.

6.3 Editor

Click the Editor tab to set up options for Code View. The code editor is displayed when Code View is selected and is used to enter and edit objects' programming code.



6.3.1 Font
From the Font drop-down list, select a font for the text in Code View. A sample of the font is shown in the "Sample Text" box. From the Point Size drop-down list, select a point size for the selected font. The "Sample Text" reflects the selected point size.

6.3.2 Colors
You can specify a background color or different colors for different parts of the programming code, as follows:

- Background
- Normal text
- Keywords
- Strings
- Symbols
- Comments
- Documentation
- Errors

You can specify that keywords, strings, symbols, comments, or errors appear in colors different from normal text to assist in finding and identifying them within the code.

Click an item to display its color in the "Sample Text" box. Double-click an item, or click **[Set Color]** to change the color. A color selector dialog box appears. Click a color in the basic colors section to select it. To create a custom color, drag the color selector in the rainbow palette to a color and then drag the slider up or down to adjust the saturation and lightness of the color; or enter the HSL or RGB of a color. Click **[OK]** to change the item to the selected color.

Use Windows Default Color
Click an item, then click **[Use Windows Default Color]** to set the color to the Windows default.

Reset All Colors to Default
Click **[Reset All Colors to Default]** to set the color of every item to the Qlarity Foundry default.

Tab Spacing
Enter the number of characters (or use the selection arrows to select the number) to be indented when you press **<Tab>**.

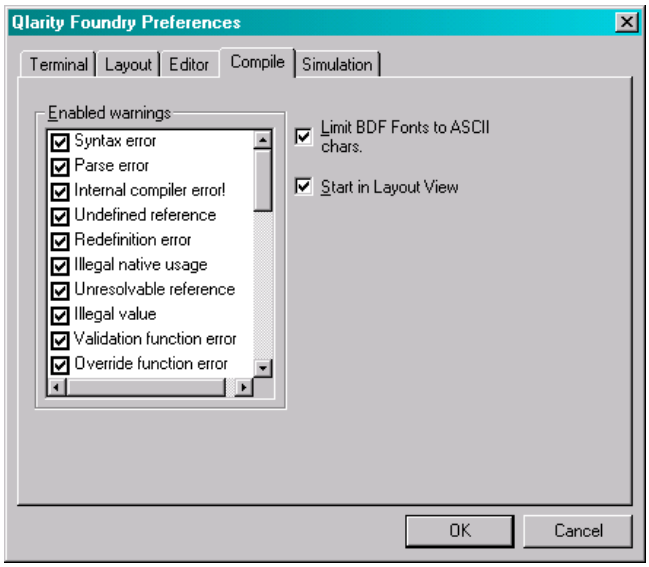
Show advanced global section
(Normally disabled) If this option is enabled, an "Advanced Code" section is added to the Object Tree beneath Templates in Code View. Advanced Code is a special section of global code. Normally, Qlarity Foundry monitors the code that you type in Code View. When you type an advanced code construct (such as a declare or define statement), Qlarity Foundry performs the action that you typed (e.g., if you type in a define statement, Qlarity Foundry removes the define statement and creates an object template). Within the Advanced Code section, this monitoring is disabled, and you can type in advanced code constructs without interference. This is only useful for advanced programmers.

6.4 Compile

Before you download a workspace to the OptoTerminal, it must be compiled into the format required for a user appli-

cation. Compiling is also required to update modifications made in Code View before they can be seen in Layout View.

Click the Compile tab to set up the options for compiling.



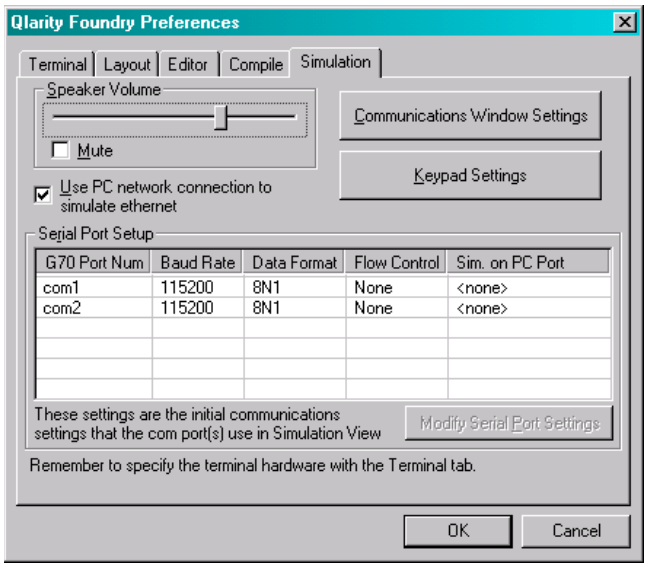
Enabled Warnings
All possible warning messages that may appear when compiling a workspace are listed. Click the checkbox to select each warning that you want displayed after a compile. When you first start programming in Qlarity Foundry you probably want to display all warnings; however, if you are an advanced user, you may find some of the less serious warnings to be unnecessary. An *error message* is displayed when an error occurs that prevents compiling. Error messages cannot be disabled.

Limit BDF Fonts to ASCII Characters
If this option is enabled, only the ASCII characters in BDF fonts are displayed. Unless you need to use characters beyond the standard ASCII character set, you should enable this option to reduce the memory requirements for BDF fonts. (OptoTerminals contain 256 ASCII characters.)

Start in Layout View
If this option is enabled, a workspace is compiled as soon as you load it. If you typically work in Layout View, you will want this feature enabled. If you work more often in Code View or your workspace has errors that haven't been corrected yet, you may want to disable it.

6.5 Simulation

Use the options in the Simulation tab to adjust or mute the sound during terminal simulation, to set up communications for the Communications (Comm View) window in Simulation mode, to set up the computer's ports to simulate the serial ports on the OptoTerminal, and to specify simulated keypad settings for applications that use an external keypad.



Speaker Volume
Click and drag the slide button to adjust the volume in Simulation mode.

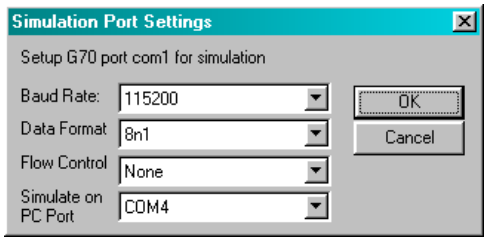
Mute
Enable this option to turn sound off in Simulation mode.

Use PC network connection to simulate Ethernet
If you select this option, your computer's network connection (if available) will be used in Simulation mode to transmit and receive data requested by the application. Disable this option if you do not want Qlarity Foundry to access the network in Simulation mode.

6.5.1 Serial Port Setup

You can set up Simulation mode to send and receive application data through a serial port or ports on your computer. Use the serial port setup to set up the computer's ports to simulate the serial ports on the OptoTerminal.

The terminal's available serial ports are listed. To specify the PC port that you want to use for the simulation and to set up the port's configuration, select the terminal port from the list, and click **[Modify Serial Port Settings]** (or double-click the port). The following dialog box is displayed.



Baud Rate
From the drop-down list, select the initial baud rate of the PC port selected in "Sim on PC Port."

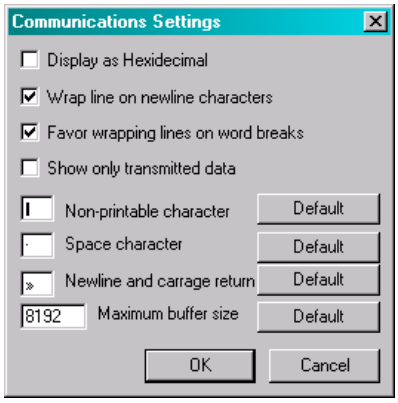
Data Format
From the drop-down list, select the data bits, parity, and stop bits of the PC port selected in "Sim on PC Port."

Flow Control
From the drop-down list, select the flow control of the PC port selected in "Sim on PC Port."

Simulate on PC Port
From the drop-down list, select the serial port on your PC that you want to use to transmit data in Simulation mode. If this option is set to **<None>**, any serial send and receive requests by the application are ignored. If you are setting up more than one port, assign a different PC serial port to each terminal serial port.

6.5.2 Communications Window Settings

Click **[Communications Window Settings]** to set up communication preferences for the Communications (Comm View) window in Simulation mode.



The default settings are generally adequate for most users, but you can change any of the following settings.

Display as hexadecimal
If this option is enabled, while in Simulation mode, data transmitted by the OptoTerminal appears in the Communications window in hex format rather than text (e.g., "ABCD" appears as "41 42 43 44").

Wrap line on newline characters
Enable this option if you want characters in the Communications window to wrap to the next line when a newline character is encountered.

Favor wrapping lines on word breaks
If "Wrap line on newline characters" is enabled, select this option to wrap to the next line after a word (rather than between characters in a word) whenever possible.

Show only transmitted data
If this option is enabled, the Communications window will display only data that is transmitted from the application program. Disable this option to show both data transmitted both received by the application.

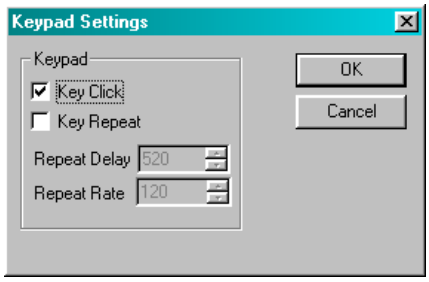
Non-printable character
Space character
Newline and carriage return
Enter the character that you want to appear on the screen in Simulation mode when a non-printable, space, or newline/carriage return character is encountered. Click **[Default]** to enter the default character.

Maximum buffer size
Enter the maximum amount of data (in bytes) to be held in the Communications window buffer. When this number of bytes is reached, data will be removed from the top of the

buffer as new data appears at the bottom. Click **[Default]** to enter the default buffer size.

6.5.3 Keypad Settings

This option only applies to applications that use an external keypad (not a keyboard). Click **[Keypad Settings]** to configure the keypad simulation.



Key Click
Select this option to turn the audible key click on in Simulation mode. If selected, you will hear a tone or "beep" when you click a key on the simulated keypad.

Key Repeat
Select this option to turn the key repeat feature on. If selected, a key entry repeats when you click and hold down the mouse button on a key on the simulated keypad.

Repeat Delay
If you selected "Key Repeat," enter the delay time (in milliseconds) that you want between when a key is pressed and when it begins to repeat automatically.

Repeat Rate
If you selected "Key Repeat," enter the time (in milliseconds) that you want between each repeat when a key begins to repeat automatically.

CHAPTER 7

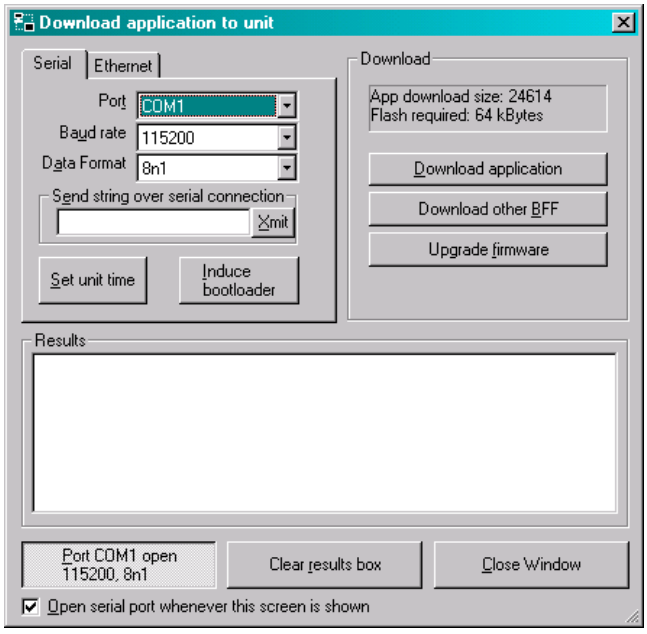
DOWNLOAD SOFTWARE TO THE TERMINAL

This chapter provides information to do the following:

- Configure Communications Port
- Download a User Application
- Download a BFF File
- Upgrade the Firmware

7.1 Configure Communications Port

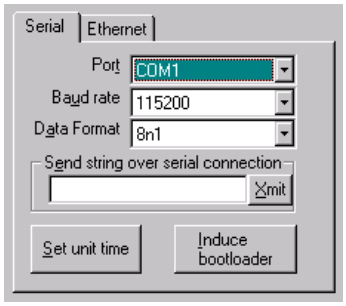
To configure the communications port used to download user applications to your OptoTerminal, select **Download Application** from the File menu. The following dialog box is displayed.



NOTE: settings must match the terminal
Your communication settings in Qlarity Foundry must match those at the OptoTerminal. For information on referencing or changing the terminal's communication settings, refer to Chapter 7 of the *OptoTerminal Hardware User's Manual*

7.1.1 Serial Port Settings

Click the Serial tab to set up a serial port.



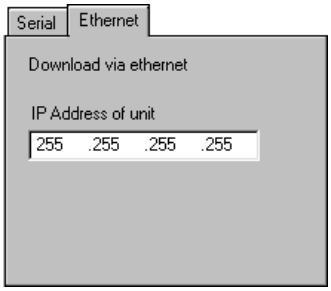
Port
From the drop-down list, select the port used to communicate with the OptoTerminal (e.g., Com1).

Baud rate
From the drop-down list, select the baud rate for the port (e.g., 115200).

Data Format
From the drop-down list, select the data format for the port (e.g., 8n1).

7.1.2 Ethernet Port Settings

Click the Ethernet tab to set up an Ethernet port.



IP Address of Unit
Enter the IP address assigned to the OptoTerminal.

7.2 Download a User Application

When you download a workspace to your OptoTerminal, it is automatically compiled into binary file format (BFF) for use as a user application on the terminal.

The following sections describe how to prepare your OptoTerminal for downloading and how to download the user application.

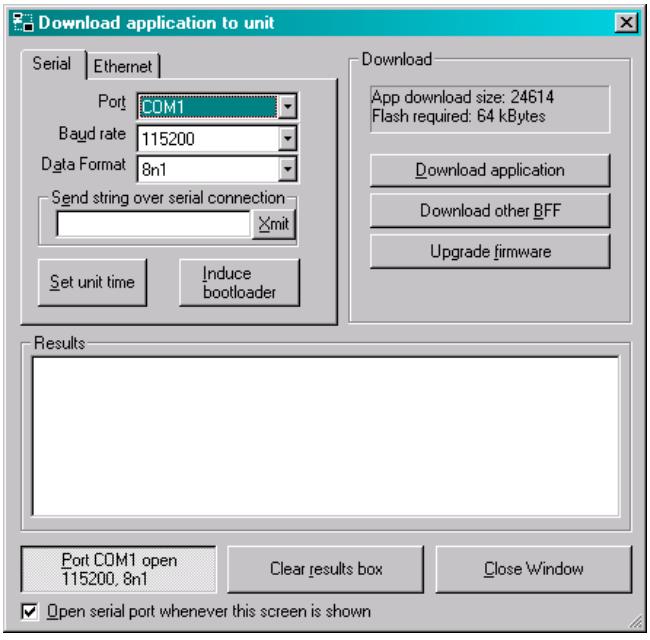
7.2.1 Prepare the Terminal for Downloading

You may need to set up the OptoTerminal for downloading using the terminal's Power On Setup utility. Refer to Chapter 7 of the *OptoTerminal Hardware User's Manual* for information on using Power On Setup.

7.2.2 Download the User Application

Do the following to download a user application to the terminal (refer to the previous section for information on preparing the terminal for download).

1. Select **Download Application** from the File menu, and the Download dialog box is displayed.



2. Click **[Download Application]** to proceed. A progress graph tracks the download, and the "Results" box displays messages indicating whether the download was successful.

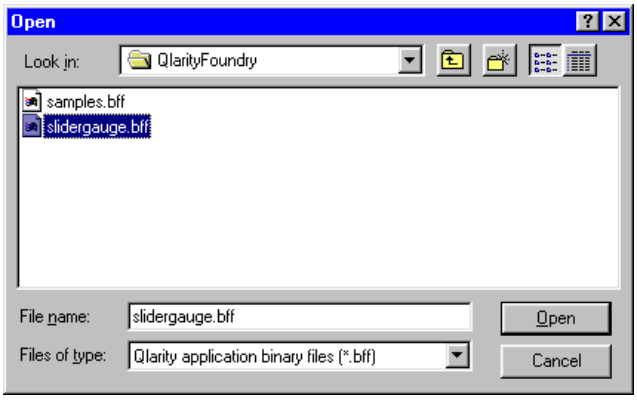
   If an error occurs, verify that your communications settings are correct. Power the terminal off and back on, and once again place the terminal in the proper mode. Repeat the download process.

NOTE: clear results
Click **[Clear Results Box]** to clear the messages in the "Results" box.

7.3 Download a BFF File

You can download a user application that is not currently loaded in Qlarity Foundry to the OptoTerminal, but the file must be in binary file format (BFF). Save a file to BFF using the "Generate BFF" function on the File menu (see section 4.6).

Click **[Download Other BFF]**, and the following dialog box is displayed.



All BFF files in the default folder are listed (change folders if necessary). Click the file that you want to download and click **[Open]**.

A progress graph tracks the download, and messages in the "Results" box indicate whether the download was successful.

7.4 Upgrade the Firmware

Use the **[Upgrade Firmware]** function to download a new version of the terminal-resident software (firmware) to the OptoTerminal.

- Refer to section 7.4.1 (below) for information on identifying the version of your current terminal firmware.

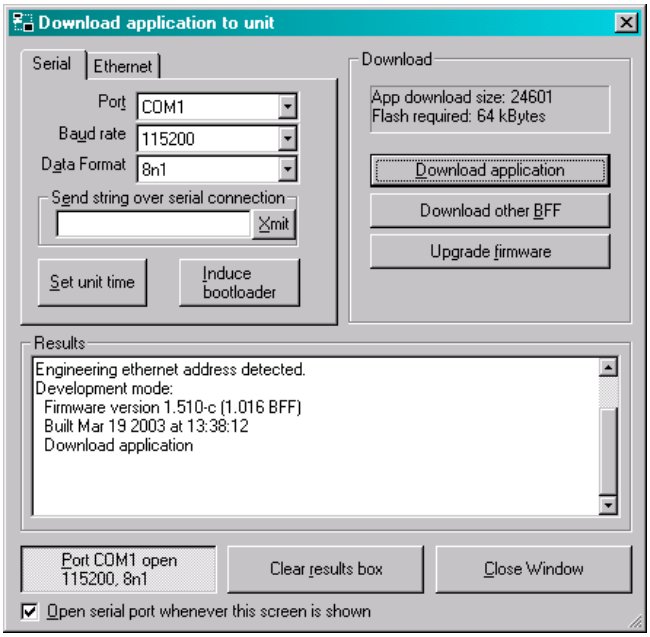
- Refer to section 7.4.2 for information on preparing your terminal for downloading firmware.
- Refer to section 7.4.3 for information on downloading the new firmware.

7.4.1 Determine Current Firmware Version

If you are not sure what version of firmware is on the OptoTerminal, do the following to determine the current version of terminal firmware.

1. Verify that the OptoTerminal is connected to your computer.
2. Put the OptoTerminal in "Develop" mode (App Mode: Develop). Refer to Chapter 7 of the *OptoTerminal Hardware User's Manual* for instructions.
3. Select **Download Application** from the File menu.
4. Power the terminal on. If it is already on, power it off and back on.

   The firmware version and date of issue are displayed in the "Results" box, as shown in the figure below.



7.4.2 Prepare Terminal for Upgrade

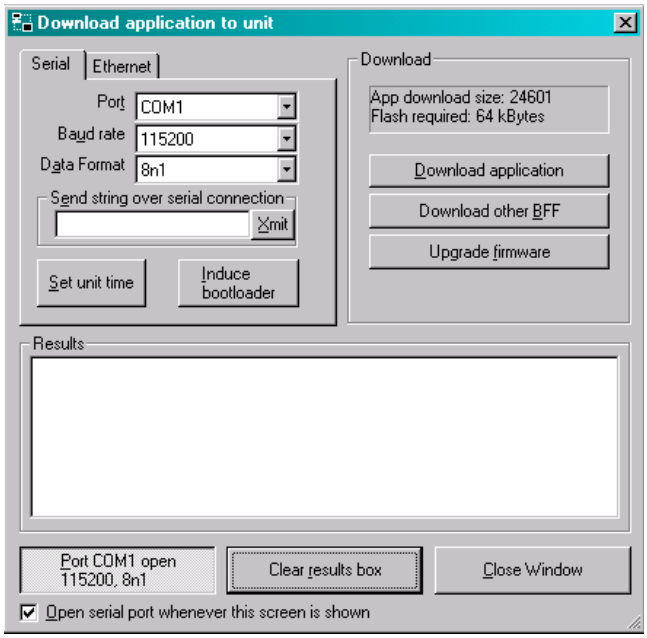
To prepare the terminal to receive new firmware using a serial or Ethernet connection, set the **App Mode** option to either **Develop** or **Download**. Refer to Chapter 7 of the *OptoTerminal Hardware User's Manual* for instructions.

NOTE: if you cannot access the terminal
If you cannot access the OptoTerminal, use the "Induce Bootloader" function (refer to section 7.4.4 for information).

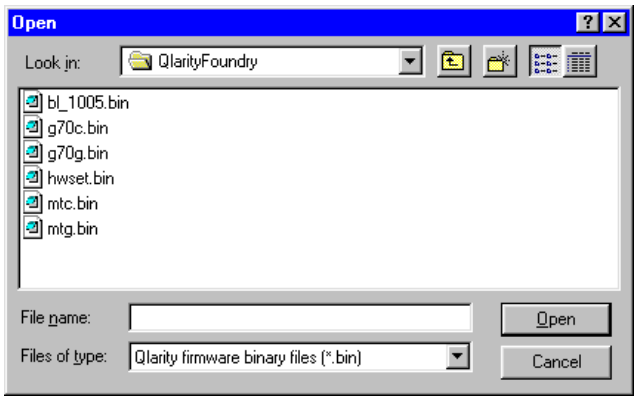
7.4.3 Download New Firmware

Do the following to download a new version of the Qlarity-based firmware.

1. Select **Download Application** from the File menu. The following dialog box appears.



2. Click **[Upgrade firmware]** to proceed, and the following dialog box is displayed.



3. Select the firmware upgrade file to be downloaded and click **[Open]**.

   A progress graph tracks the download. Messages in the "Results" box indicate whether the download was successful. When finished, the new firmware version is shown in the "Results" box.

7.4.4 Induce Bootloader

If the firmware on your OptoTerminal has become corrupt and you cannot run the Power On Setup utility, use the "Induce Bootloader" function to upgrade (or reinstall) the firmware.

NOTE: Ethernet connection
If your terminal is connected to an Ethernet port, you will need to change it to a serial port to perform this function.

Do the following to download a new version of the OptoTerminal firmware.

1. Select **Download Application** from the File menu.

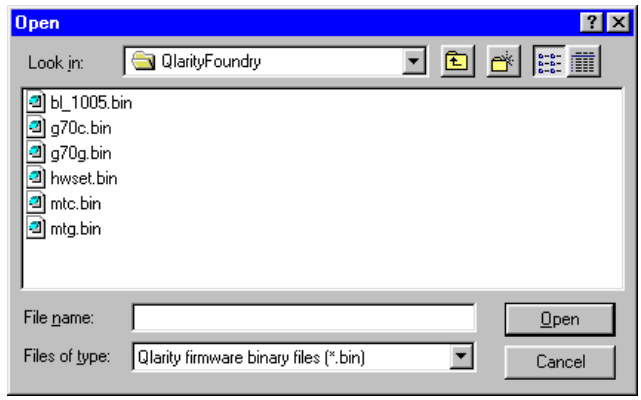
2. Click **[Induce Bootloader]**, and an "Inducing Bootloader" message is displayed in the "Results" box. After a time the following message is displayed:

   ...Bootloader induction terminated

   Followed by the message:

   NOTICE--received command from host to enter bootloader
   Entering Opto 22 Bootloader V1.014!

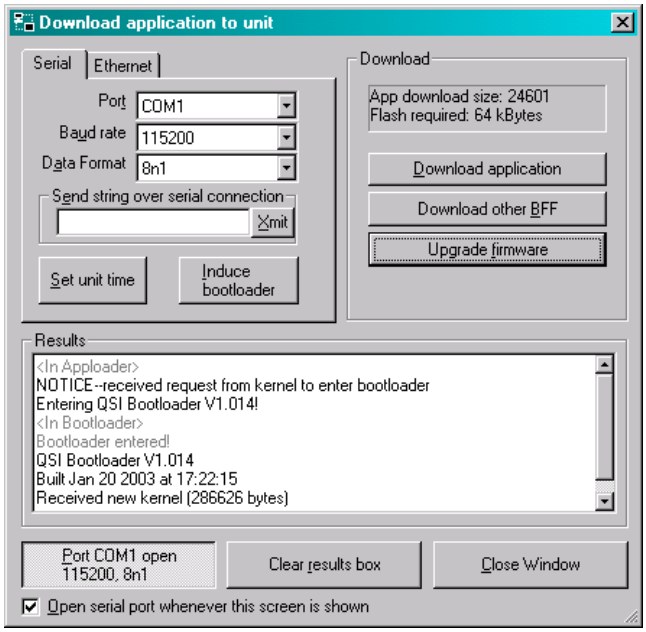
3. Click **[Upgrade firmware]** to proceed, and the following dialog box is displayed.



4. Select the firmware software file to be downloaded and click **[Open]**.

   A progress graph tracks the download. Messages in the "Results" box indicate whether the download was successful.

When finished, the new firmware version is shown in the "Results" box.



7.4.5 Set Unit Time

Use the Set Unit Time function to set the real-time clock on the OptoTerminal, as follows.

1. The terminal must be in "download" or "develop" mode. If necessary, set up the OptoTerminal for downloading using the terminal's Power On Setup utility. Refer to Chapter 7 of the *OptoTerminal Hardware User's Manual* for instructions.

NOTE: time set through serial port
The time is set through the serial port connection. If your terminal is connected to an Ethernet port, you will need to change it to a serial port to perform this function.

2. Click **Set Unit Time** (bottom of the Serial tab).

   The message, "<Setting Time>" is displayed in the "Results" box. If no further message is displayed, the terminal's time has been set. If an error occurs, it is displayed in the "Results" box, as shown below.

CHAPTER 8

BASIC DESIGN

You can use Qlarity Foundry to design a user application using library objects and/or existing object templates. This "basic" design requires no special programming skills.

This chapter provides instructions for using Qlarity Foundry to design a user application using only the objects and templates provided, as well as information on the basic concepts of the Qlarity programming language.

All Qlarity Foundry users should review and become familiar with the information in this chapter. If you want to learn more about using Qlarity Foundry and Qlarity programming, you can then proceed to Chapter 9, "Intermediate Design" and Chapter 10, "Advanced Design."

This chapter provides the following information:

Prepare Qlarity Foundry for Application Design

- Basic Design Layout
- Simulate the Terminal Display
- Drawing Aids (Zoom, Grid, Snap, etc.)
- Add/Remove Resources
- Add/Remove Libraries

Understanding Qlarity for Basic Design

- Workspaces and User Applications
- Qlarity Objects
- Parent/Child Relationships
- Z-Order
- Events and Messaging
- Enabled/Disabled Objects

Design a User Application

- Add an Object Instance
- Move, Resize and Reorder Objects
- Change Object Properties

Event Builder

- Overview of Event Builder Steps
- Event Builder Dialog Box
- Select and Configure Actions
- Load Event Builder Sample Workspace
- Qlarity Code and Event Builder
- Troubleshooting

Communication Objects

- Serial Objects
- Ethernet Objects
- Receive Data

Test the User Application

Save and Compile a Workspace

Download a User Application

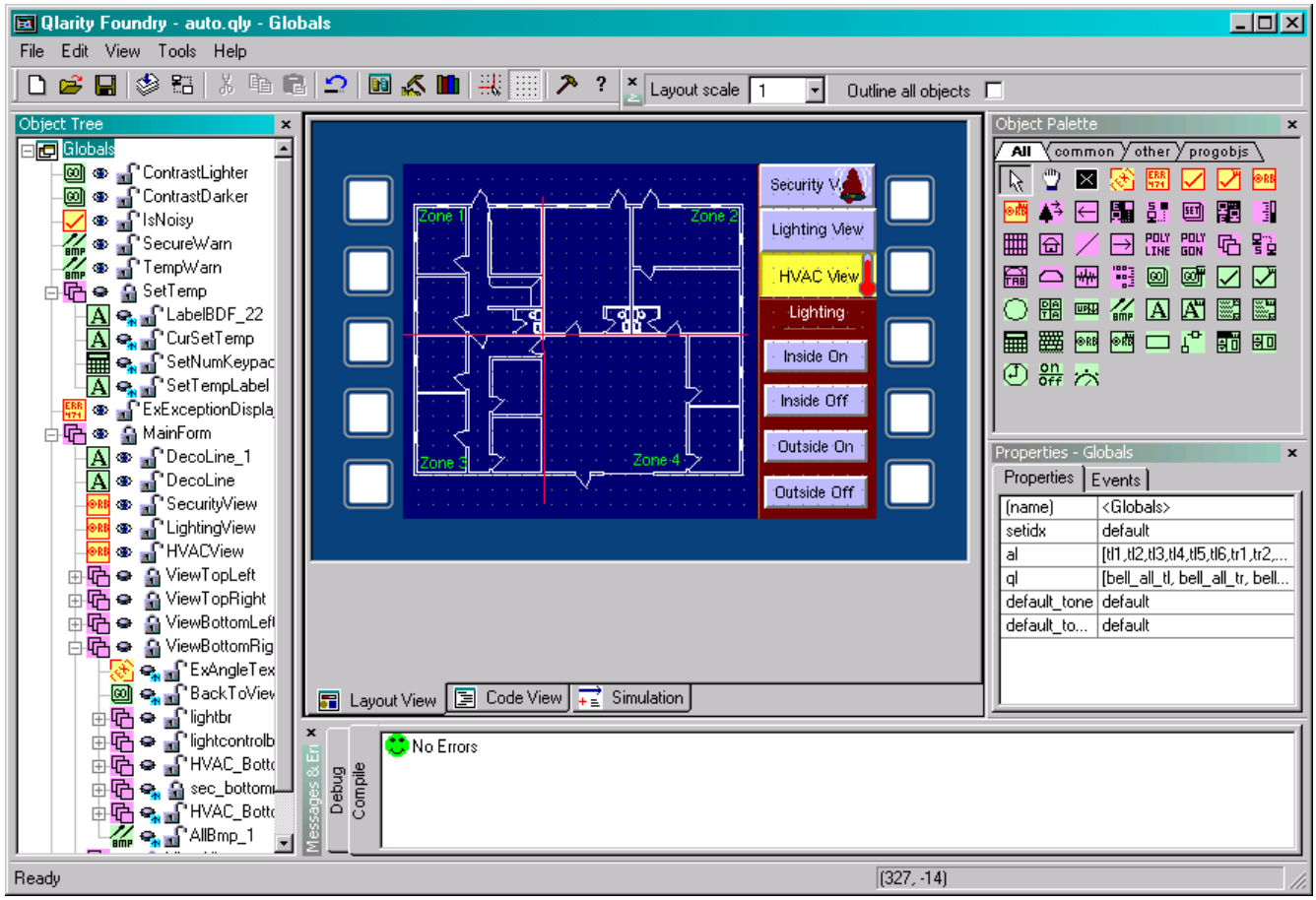
8.1 Prepare Qlarity Foundry for Application Design

This section provides information on how to set up Qlarity Foundry before you design a user application. Begin by opening Qlarity Foundry and starting a new workspace. Refer to Chapter 3, "Getting Started" for information.

8.1.1 Basic Design Layout

You work only in Layout View to create a basic design. You should keep the following Qlarity Foundry windows open, as you will use all of them:

- Toolbar
- Object Properties
- Object Palette
- Object Tree



All four windows are open by default. If one is inadvertently closed, you can open it from the View menu.

NOTE: workspace appearance variations
In the illustration above and elsewhere in this chapter, the objects in the Object Palette and Object Tree may differ from what you see in Qlarity Foundry. Object names may also be different. This is the result of ongoing improvements to Qlarity Foundry libraries.

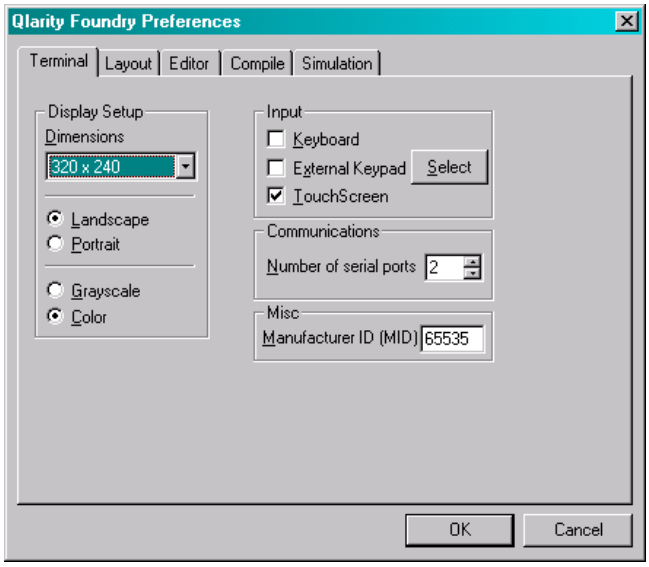
8.1.2 Simulate the Terminal Display

Qlarity Foundry requires information about the display configuration of your OptoTerminal to simulate it accurately in the work area and in Simulation mode. The settings required for this include the following:

- Display setup (including landscape/portrait and gray-scale/color)
- Valid input devices (type of key input used with your terminal)
- Touch key legend (border around the edge of the touch screen for a Opto 22 standard touch key legend—enable if you are using a touch key legend; disable if you are not.)
- Simulation setup (volume in Simulation mode and Communications window defaults) The Communications window simulates the data received and sent via the communications port.

Click on the toolbar, or select **Settings** from the View menu.

The Qlarity Foundry Preferences dialog box is displayed.

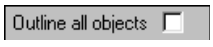


Tabs are available to define your OptoTerminal display, to define the layout of the Qlarity Foundry work area (which includes the Opto 22 standard touch key legend setting), to set up compile defaults, and to set up the Simulation display. Refer to Chapter 6, "Qlarity Foundry Preferences" for more information.

8.1.3 Drawing Aids

When arranging the graphical objects on the terminal display, proper alignment is important. Qlarity Foundry provides the following tools to assist you with object layout.

To change the scale of the work area, click the drop-down arrow and select the scale from the drop-down list.



Select this option to draw a line around all defined objects. This is useful when you want to know the exact border location of each object, or when one or more objects was accidently moved out of the work area.

Grid
You can display a grid over the work area to help you more accurately place and align objects. You can turn the grid off and on and change the spacing and color of the grid lines.

Snap to Grid
If enabled, this feature forces objects to "snap" to the nearest grid line when added or moved. The top left corner of an object's rectangle snaps to the nearest horizontal and vertical grid lines when you release the mouse button.

To customize the grid and enable "Snap to Grid," click on the toolbar, or select **Settings** from the View menu. Click the "Layout" tab, and edit the follow settings as needed (refer to section 6.2 for more information).

- Show Grid (show/hide grid)
- Snap to Grid (enable/disable snap to grid)
- Grid Spacing (set the spacing of the grid lines)
- Colors, Grid (set the color of the grid lines)

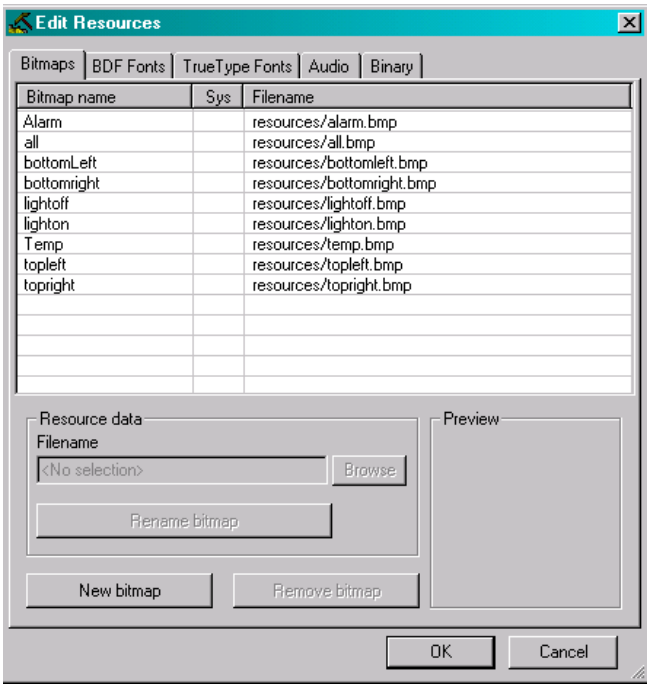
8.1.4 Add/Remove Resources

Resources include all bitmap images, fonts, audio files, and binary data files available in a workspace and, eventually, in the user application. You must add resources to a workspace before you can use them in object instances.

Use the Edit Resources option to add resources to your workspace or to change or remove the resources used. Because resources become part of the user application when it is compiled, you should delete any unnecessary resources.

NOTE: default bitmaps and fonts
When you create a new workspace, you have the option to add default bitmap images and fonts to your workspace. Some objects may require at least one bitmap, BDF font, or TT font resource before the object will function properly. When you have finished creating a workspace, however, you may want to remove any unused default bitmaps and fonts before you download the user application to the OptoTerminal to conserve the terminal's flash and RAM memory.

Click on the toolbar, or select **Edit Resources** from the Edit menu, and the Edit Resources dialog box is displayed.



All resources used in a workspace are managed from this dialog box. Select the tab for the resource type you want to manage. You can add a new resource, rename an existing resource, change the file for an existing resource name, or remove a resource from the workspace. Refer to section 5.2 for information on editing resources.

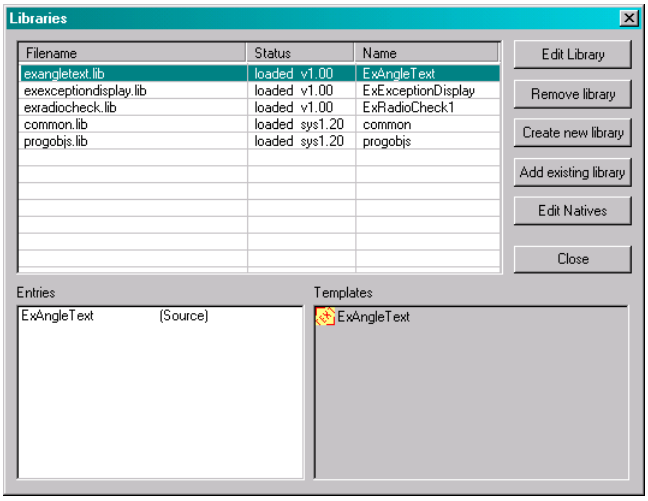
8.1.5 Add/Remove Libraries

A library is a file that contains predefined object templates that you can use to create user applications. Library object templates are programmed to be flexible. This means that you can modify them, without additional programming, to meet most requirements.

In basic design, libraries are your primary source of new objects. When you create a new workspace, your selections in the New Qlarity Project dialog box determine which libraries are automatically added to the new workspace. You can also add or remove libraries after you open the workspace .

NOTE: library object templates
Object templates in libraries are not listed in the Object Tree under the Templates heading. Only object templates created in the workspace are listed under this heading.

To add a library to or remove a library from a workspace, click on the toolbar, or select **Edit Libraries** on the Edit menu. A dialog box similar to the following is displayed.



Click **[Add Library]** to add a new library to the workspace. Click **[Remove Library]** to remove the selected library. For additional information on libraries, refer to section 5.3.

8.1.5.1 Libraries Provided with Qlarity Foundry

Qlarity Foundry includes libraries of object templates that allow you to create interactive, complex user applications without entering any programming code. After you add one or more of these libraries to a workspace, you can change the object properties to customize the object instances in your user application. Two of the libraries provided with Qlarity Foundry that are used at the basic design level are:

- Common library (common.lib)
- Programmable objects library (progobjs.lib)

These libraries include the following types of objects:

- Container objects with tabs or buttons to switch from one screen to the next (Common library).
- Area objects with event functionality (Programmable objects library). Refer to section 8.4, "Event Builder" for information on configuring objects to respond to events.
- Communication objects (Common library). Refer to section 8.4, "Event Builder" and section 8.5, "Communication Objects" for information on how to use communication objects to send data to another device through the OptoTerminal's serial or Ethernet port.

The Qlarity Foundry libraries are continually being added to and improved. You can obtain the latest libraries at Opto 22s Web site *www.opto22.com*. Also on the Web site you will find descriptions of the objects included in the libraries.

8.2 Understanding Qlarity for Basic Design

Before you can effectively design a user application, you need to understand some basic Qlarity concepts, which are explained in the following sections.

8.2.1 Workspaces and User Applications

A *workspace* is a Windows-based file created in Qlarity Foundry that is compiled into a *user application*. Workspace files are stored as plain text and typically have a *.qly* file name extension. User application files have a *.bff* extension.

In Qlarity Foundry, you use a workspace to define functions for the terminal; at the OptoTerminal, you use a user application to perform the functions.

When a workspace is compiled into a user application, the Qlarity interpreter initializes the user application in Qlarity Foundry so you can see the results. You can then download the user application to the OptoTerminal.

8.2.2 Qlarity Objects

A user application is constructed of Qlarity-programmed *objects.* An *object instance* is an occurrence of an object in the user application. An *object template* contains the programming code that defines an object. When you create a basic design, you will use objects from existing libraries (several libraries are provided with Qlarity Foundry).

There are three types of objects, as follows. All three types of objects can be found in Opto 22 libraries.

Non-drawable object:
An object that serves a function not related to the display (e.g., key definition, communications, etc.).

Area object:
An object that can be represented visually on the terminal display. An area object might be any of the following:

- Bitmap (an imported bitmap image)
- Lines, circles or rectangles (may include custom borders and fill colors)
- Text (labels, headings, or a field where text may be typed or displayed)

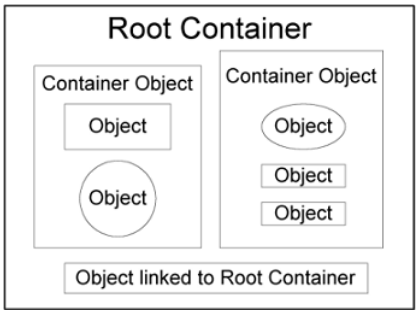
Resources (bitmaps and fonts) are required for text and bitmap objects.

Container object:
An area object that contains other objects (such as the Screen object in the common.lib library). A container object may occupy part of the terminal display or the full display area. You use container objects to define an association between objects and to indicate how they are displayed. All objects in a container process information as a group. Containers are often stacked and identified with tabs.

8.2.3 Parent/Child Relationships

Qlarity allows you to group objects together. How objects are grouped determines how information is processed (messaging) and how the terminal screen redraws. The complexity of your user application determines how much you need to group objects. Grouped objects have a "parent/child" relationship, with the parent always a container object and the objects inside the container the children.

Each user application starts with a "root container." Though invisible, it is the container in which you place all other objects. The following illustration shows a root container with several objects in it.

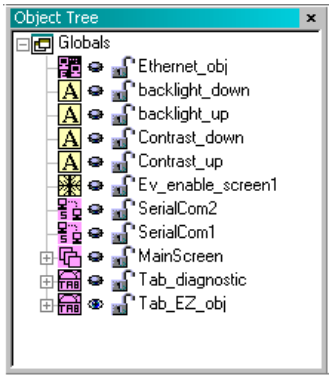


In this illustration, two container objects plus another object are linked to the root container. In addition, each container object has objects linked to it, which are referred to as its children.

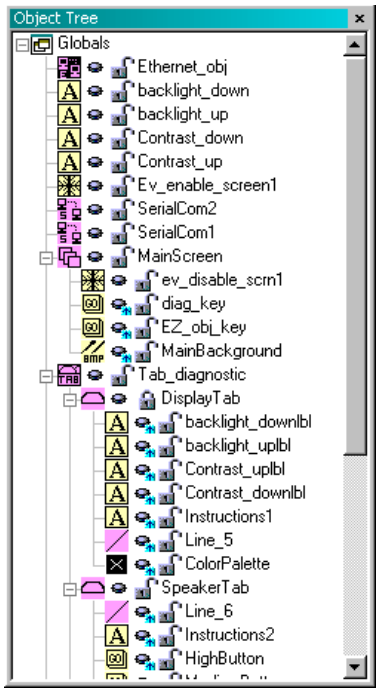
In Qlarity Foundry, when you add a container object (e.g., tab, screen, etc.) to your work area and then place new

objects inside the container, the objects automatically become children of the parent container object. If an object is not placed inside a container object, it is linked to the root container by default.

When you collapse the Object Tree, you can quickly identify all objects in the root container. In the following illustration, the first two objects, EZBackButton_1 and EZBackButton_2 are area objects that have been placed in the root container. The **[+]** next to EZScreen_3, EZScreen_1 and EZScreen_2 indicate that they are container objects with other objects in them (parents with children).



Click **[+]** next to a "parent" container object to see its "children," as shown below.



A parent object can also be a child (a container object with children inside another container object). "HardwareNew" and "SoftwareNew" in the illustration are examples of parent objects within other parent objects.

8.2.4 Z-Order

Z-order is the order in which objects are layered. The Z-order determines how objects are displayed (whether they are behind or in front of another object or objects) and the order in which information is processed (messaging). In a Qlarity Foundry workspace, the Object Tree lists objects in their Z-order, with those listed first receiving the highest display and messaging priority.

To change an object's Z-order, click and drag the object name to a different position in the Object Tree. To learn more about how parent/child relationships and Z-order affect user applications, read the next section.

8.2.5 Events and Messaging

Qlarity is an event-driven programming language. An event is any type of input, such as the following:

- A keyboard key press or release
- A touch screen press or release
- A change in a value (increment or decrement)
- A timer action
- Data received through the serial port

When an event occurs, a Qlarity user application responds with all appropriate and specified actions.

The system generates a message indicating that the event has occurred and what it is. The terminal's message handling system determines which object or objects get the message and in what order the messages are processed.

The message is first passed to the root container, which passes the message to each of its children, beginning with the front-most object (highest Z-order). When the message is passed to a container object, this object first handles the message then passes it to each of its children beginning with the front-most object in the container. The terminal processes the message to completion.

8.2.6 Enabled/Disabled Objects

An object instance can be enabled or disabled. On the OptoTerminal, only enabled objects are displayed. Disabled objects still reside in the user application, but they cannot receive or send most messages, nor are they displayed on the terminal. If a container object is disabled, it cannot pass messages to its children.

In Qlarity Foundry, all objects are displayed in the work area (by default) whether they are enabled or disabled. However, you can select the "View Only Enabled Objects" option on the Tools menu to hide disabled objects.

8.3 Design a User Application

This section provides instructions for adding and working with object instances in a workspace to create a user application. For information on adding, opening, and saving a workspace, refer to Chapter 4, "Workspaces."

8.3.1 Add an Object Instance

You can add object instances to your workspace from the Object Palette or from the Object Tree shortcut menu, as explained in the following sections.

8.3.1.1 Add an Object From the Object Palette

The Object Palette contains a tab for each library in the workspace, as well as an "All" tab and an "Other" tab. Each tab has icons for all of the object templates in the library. Click a tab to select an object from the specified library.

The "All" tab contains icons for all objects available to the workspace (from all of the libraries in the workspace). The background color of each icon identifies the library in which the object can be found. The "Other" tab contains objects from various small libraries.

Refer to section 3.11, "Object Palette" for more information on the Object Palette.

Do the following to add a new object instance from the palette.

1. Click the icon of the object you want to add.

2. Move the mouse pointer to the work area. The pointer changes to a cross hair.

3. Click and hold the mouse button, and drag the mouse to draw a rectangle in the work area. Then release the mouse button.

   Some objects have a default starting size and shape, so the size of the rectangle doesn't matter, only its position in the work area. The size of other objects, such as a line or rectangle, is initially determined by the size of the rectangle you draw.

   If you are adding a non-drawable object, the location in the work area doesn't matter since the object is not part of the display. You may want to use the shortcut menu to add a non-drawable object in its default position (see section 8.3.1.2, "Add an Object From the Shortcut Menu" for information. Refer to Appendix B for more information on non-drawable objects.

NOTE: add multiple instances of the same object type
To add more than one instance of the same type of object, press and hold the **<Shift>** key when you click the object icon. The object type remains selected, and you can just click and drag in the workspace to add objects until you release the **<Shift>** key.

4. After you add an object, it remains selected and its object properties are displayed. Click the "Name" property, and type a name for the new object (type over the default name). Each object in the workspace must have a unique name. An object name has no size limitation but must start with a letter. A name cannot contain spaces but may use the underline character (_). The percent (%), pound (#) and dollar sign ($) symbols can be used at the end of the name.

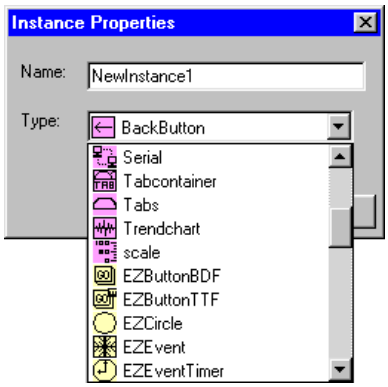
Refer to section 8.3.2 for information on moving, resizing and changing the order of objects.

8.3.1.2 Add an Object From the Shortcut Menu

Do the following to add a new object instance from the Object Tree shortcut menu.

1. Right-click anywhere in the Object Tree to display the shortcut menu.

2. Click **New Object Instance**. The Instance Properties dialog box is opened. All available objects are listed in the drop-down list at the "Type" field, as shown below.



3. At the "Name" field, type a name for the new object (type over the default name). Each object in the workspace must have a unique name. An object name has no size limitation but must start with a letter. A name cannot contain spaces but may use the underline character (_). The percent (%), pound (#) and dollar sign ($) symbols can be used at the end of the name.

4. At the "Type" field, click the drop-down list, and select the type of object that you want to add. After you select an object, it is added to the work area at its default location and is listed in the Object Tree.

The newly added object remains selected and its object properties are displayed. You can edit the properties as required.

8.3.2 Move, Resize and Reorder Objects

To select an object to move, resize or reorder, click on it. The selected object is outlined with a colored box. You can only select one object at a time. You cannot move or resize a locked object.

Press <**Tab**> to move forward from object to object in Z-order. Press <**Shift**>+<**Tab**> to move backwards.

NOTE: Object Properties
When you press <**Tab**>, Object Properties changes to match the newly selected object.

8.3.2.1 Move an Object

To move an object, do one of the following:

- Use the mouse. Click in the center of the object, hold down the mouse button, and drag the object to a different position. You can hold down the <**Shift**> key while moving an object to limit its movement to horizontal, vertical, or a 45° angle.
- Use Object Properties. Enter a new x and/or y position integer in Object Properties.

8.3.2.2 Resize an Object

To resize an object, do one of the following:

- Use the mouse. If the object has "resize grips" (sizing handles), you can click and drag any grip to resize the object. Click and drag a corner grip to maintain the height/width proportions while resizing.

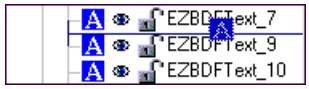
- Use Object Properties. Change the height and/or width integer in Object Properties.

8.3.2.3 Change the Order of Objects

You can change the Z-order of an object or the parent/child relationship simply by changing the object's position in the Object Tree.

To move an object to a different location in the Object Tree, click and drag it to the desired position. The icon for the object is dragged to the new location.

A horizontal placement line, as shown in the illustration below, appears as you drag the object to indicate the new position at which the object will be placed when you release the mouse button.



A box appears around a container object as you drag another object over it. If you release the mouse button when the box appears, the object you are dragging is placed in the container object.

If you move a parent object (a container object with children), the children move with it. You can place a container object inside another container object.

To change the Z-order of an object within its family (without moving it to a different parent), right-click on the object (either in the Object Tree or in the work area), and select one of the following options from the shortcut menu.

Forward one
Move the object forward one position in the Z-order (one position up the Object Tree).

Back one
Move the object back one position in the Z-order (one position down the Object Tree).

Bring to front
Move the object to the front of the Z-order (top of the list of children or top of the Object Tree).

Send to back
Move the object to the back of the Z-order (bottom of the list of children or bottom of the Object Tree).

8.3.3 Change Object Properties

Each object has *properties* that help define the object instance. Click an object in the work area, or click an object name in the Object Tree, to display Object Properties. If Object Properties is not displayed, pull down the View menu, and click **Object Properties**.

The default properties for an object instance are defined in the object template from which the object instance was added. You can change any of the default properties for an object instance to create a unique object that meets your requirements.

Properties that are common to almost all objects include:

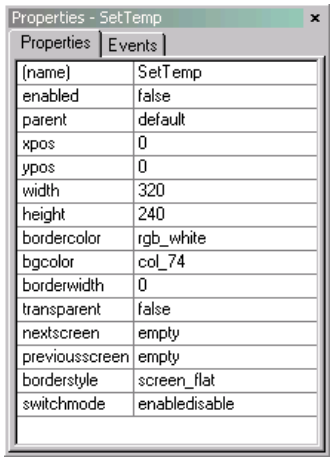
- Object name
- Enable/disabled status
- Parent specification

A wide range of additional properties vary depending on the type of object. For example, some typical properties for area objects include:

- Size ("height" and "width")
- Alignment ("xalign" and "yalign")
- Color (for borders, fill, foreground, background, font, etc.)
- Font (for a text area object)
- Position in display ("xpos" and "ypos," where both positions start at the top left corner of the parent object. The "xpos" value increases right; "ypos" increases down.)

For specific information about any Opto 22 library object, open Object Documentation (**<F1>**).

The following illustration shows an example of Object Properties for an area object.



The first column lists the name of each property. Property names are defined in the object template. The middle column (optionally displayed; see section 6.2, "Layout") lists the data type of each property (for information only). The property settings, which you can change, are listed in the third column.

Click the property that you want to change. Enter the new property setting in the third column. You can enter new property settings by typing the new information (type over the default setting) or by selecting from a drop-down list or dialog box.

In the following illustration, the property "ezimage" is selected. This is the name of the bitmap image used in the object instance. The drop-down list shows all bitmap images available for use in the object instance. A different image can be selected by clicking it.

If ▼ appears in a property field when you select the property, a drop-down list of property options is available. Click ▼ to display the drop-down list, and then click the option you want to use.

If the property is a workspace resource (bitmap image, font, sound, or binary data) or a color, a Select button (...) is displayed next to the drop-down arrow. If the resource or color you want is not in the drop-down list, click ... to open the Qlarity Resources dialog box or the Select Color palette. Select a resource or color, and click **[OK]** to add it to the workspace and to the property's drop-down list. Select the resource or color from the drop-down list to place it in the property. Refer to section 5.2, "Edit Resources" for information on the Qlarity Resources dialog box. Refer to section 8.3.3.1, "Select Color" for information on selecting a color.

NOTE: selecting a parent object
If you want an object instance to be the child of a parent object (placed in a container object), from the "parent" property drop-down list, select a container object, and the object instance is automatically moved and attached to the selected object in the Object Tree. To move an object to the root container, click "default" in the drop-down list.

8.3.3.1 Select Color

If color is applicable for an object property (e.g., a background or text color), a Select button (...) appears next to the drop-down arrow for the property. Click ... to open the Select Color palette, as shown below.

The current color of the property is displayed beneath the palette in both color and gray scale, as well as appearing in the color palette with a box around it.

Click a color in the palette to select a different color for the property.

If you want to match a color from another object in the workspace, click , and an image of the workspace is displayed in front of the palette. Click in the area of the workspace that has the color you want to match.

The color of the area you clicked is shown in the "Sample" box. In addition, the "4X" box displays the area magnified four times.

You can click a color in the "4X" box to select it. This is useful if you want to identify a color in a congested area of the display. You can also click in the "4X" box, and drag the mouse to shift the displayed area slightly.

Click **[OK]** to close the workspace image, and the color you selected is shown in the Select Color palette.

Click **[OK]** at the Select Color palette to insert the selected color in the property.

8.4 Event Builder

Qlarity is an event-driven programming language. An event is any type of input, such as a key or button press, a value change, a timer action, data received through the serial port, and so on. When an event occurs, a Qlarity user application responds with an action.

Event Builder is a tool in Qlarity Foundry to help you add events to your Qlarity user applications. Event Builder allows you to assign actions to events without any programming knowledge.

Using the options in the Event Builder dialog box, you can link one or more actions to an event, even when the action involves several objects or includes one or more serial objects. You assign an action to an object and its properties from drop-down lists.

The type of object you select determines the type of event or events that can be performed by the object. For example, a button will have a "click" event, and a serial object will have "data receive" and "send data" events. Keep this in mind when you are adding objects to your workspace. The Object Documentation Help file (**<F1>**) provides information about all object templates in the Opto 22 libraries, including events and actions that can be performed by the object type.

8.4.1 Overview of Event Builder Steps

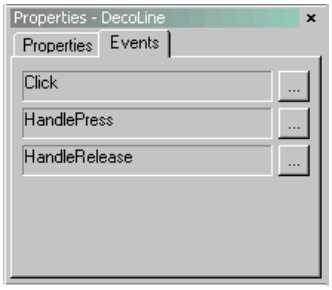
To design a user application using Event Builder, you will generally take the following steps:

- Decide what you want your Qlarity application to do.
- Determine which objects you need. Some objects will be used for events; others will perform the actions associated with the events. Place all of the objects in the workspace.
- Outline each event/action association (e.g., when a button is pressed, an action or actions occur).
- Use Event Builder to assign actions to each event.

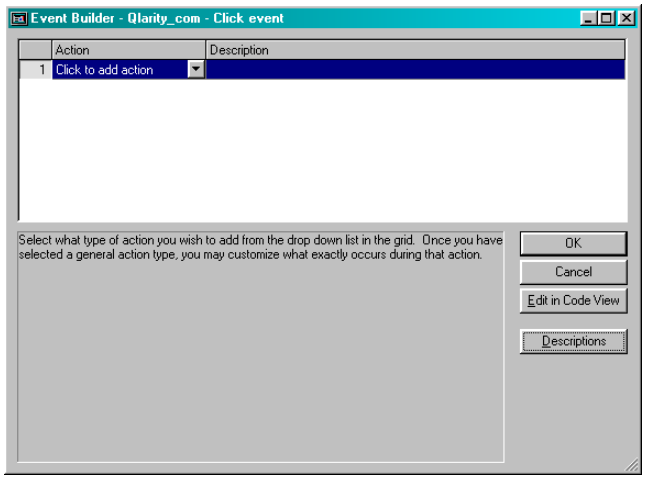
8.4.2 Event Builder Dialog Box

In the Layout View, select an object in the workspace for which you want to set up an event. The properties and events associated with the object are displayed in Object Properties.

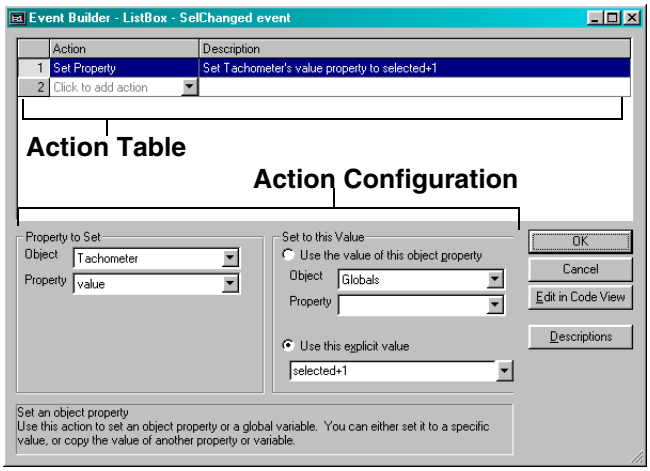
Click the **Events** tab to see the events associated with the selected object.



Click ... next to an event to open the Event Builder dialog box to assign actions to the event.


Event Builder - Qlarity_com - Click event
Action
Description
1
Click to add action
Select what type of action you wish to add from the drop down list in the grid. Once you have selected a general action type, you may customize what exactly occurs during that action.
OK
Cancel
Edit in Code View
Descriptions

The Event Builder dialog box contains the *action table* that lists all actions assigned to the selected event and the *action configuration* section that contains options to set up each action.



Action Table
The action table lists all actions configured for the selected event. Each action is on a separate line in the table.

To modify an existing action, click the action in the table. To add a new action, click the last line in the table (labeled "Click to add action"), and select the action from the drop-down list. Then modify or set up the selected action in the action configuration area.

Action Configuration
After you select an action in the action table, applicable options are displayed in the action configuration area to set up the action.

Click **[OK]** to accept the event as currently configured. If any configuration contains an error, a message is displayed.

NOTE: EZObjects
Qlarity Foundry also includes an EZObjects library (ezlib) that contains object templates with event functionality programmed into them. However, with EZObjects, you are limited to the objects in the library. Note that EZObjects cannot be configured in Event Builder. Refer to Appendix B, "EZObjects Library" for more information.

Edit in Code View
Click **[Edit in Code View]** to view the event's actions as programming code. If you understand the Qlarity programming language, you may use Code View to work with programming code. You will generally only work in Code View if you are customizing actions or creating your own actions. If you are learning Qlarity programming, it may also be helpful to view the code to understand how an event is programmed.

Descriptions/Code Preview
Click **[Descriptions]** to display the Qlarity programming code for the actions in the "Description" column of the action table. The button label changes to **[Code Preview]**. Click **[Code Preview]** to switch back to a description of each action in the table.

8.4.3 Select and Configure Actions

A sample workspace, *eventbuilder.qly*, containing events created using Event Builder is included with Qlarity Foundry. For information on loading and using the sample workspace, refer to section 8.4.4, "Load Event Builder Sample Workspace." The sample workspace is used in the images and examples in this section. You may find it helpful to load the sample workspace and use it to view examples of the actions described in this section.

8.4.3.1 Select Actions

To select actions for an object's events, do the following:

1. In the open workspace, (e.g., *eventbuilder.qly*) select the object for which you want to set up an event (e.g., Tank_slider).

2. In Object Properties, click the **Events** tab. All events associated with the object are shown. For example, the Tank_slider object in *eventbuilder.qly* has one event, "Change."

3. Click ... next to the event to open the Event Builder dialog box. The action table lists all actions that have been configured for the event.

	Action	Description
1	Serial Transmit	Transmit: "Sends tank level out COM1 to your controller " out COM1
2	Set Property	Set Slider_label's caption property to Str(Tank_slider.value)
3	Set Property	Set Tank_gauge's value property to Tank_slider.value
4	Play Note	Play note #43 for 12 ms.
5	Click to add action	

The last line of the action table always contains a drop-down list that you use to add a new action to the event.

4. Click anywhere on the line to open the drop-down list, and select an action. The action and its description are added to the table.

   If you want to add a new action in the middle of the table, or remove an action from the table, right-click in the action line. A shortcut menu is displayed with options to insert or remove actions.

5. Refer to the next section for information on each type of action and how to configure them.

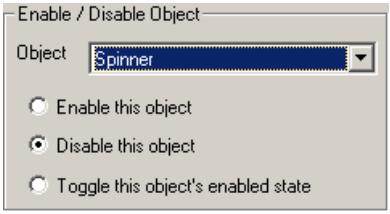
8.4.3.2 Configure Actions

The action drop-down list contains the following actions:

- Enable/Disable Object
- Set Property
- Serial Transmit
- Play Note
- Set Contrast
- Set Backlight
- Custom Action

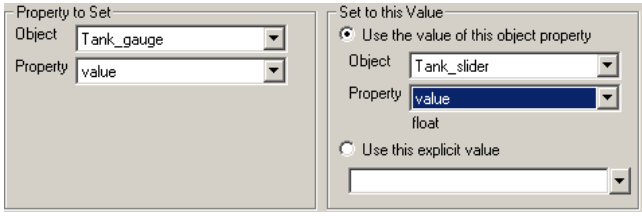
After you select an action, options appear in the action configuration area of the Event Builder dialog box to configure the action. Refer to the following sections for information on each action's configuration.

Enable/Disable Object
This action enables or disables the selected object. When an object is disabled, it is not shown on the terminal display and will not respond to most events. In the following action example, when the event occurs, a spinner object is disabled. To enable the spinner object, a second event and action are required.



A common use of this type of action is to change screens in response to a button press. For example, in the click event of a ButtonBDF object, there might be two object enable/disable actions, one to disable the currently displayed screen and another to enable the next screen.

Set Property
This action sets the value of an object property or global variable. You can use Set Property, for example, to change the value of a property in another object. The property to set is the action that results from the event.



In the "Property to Set" section, select the target object (object to be changed when the event takes place, such as moving a gauge) from the "Object" drop-down list. From the "Property" drop-down list, select the target property (property to be changed by the action).

Use one of the following options to complete the configuration:

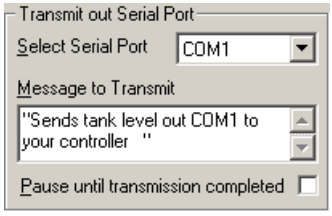
Set to this value
Select **Use the value of this object property**. Then select the source object from the "Object" drop-down list and the source property from the "Property" drop-down list. In the example above, the target and source properties both use the "value" property, but any compatible property can be used. For example, if the target object is a label, you might select the "caption" property.

Use this explicit value
For some properties, you may need to enter or select an explicit value. For example, if you want your own message displayed when a button is pressed, enter the message in this text box. A drop-down list may appear with available values, depending on the object selected.

Some objects may require additional configuration, such as selecting a color. If additional configuration is available for the selected object, a ... button appears next to the text box.

NOTE: some objects are not compatible
Some object properties are not compatible. If you select an incompatible value for the target property, an error is generated when you click **[OK]** and close the Event Builder dialog box.

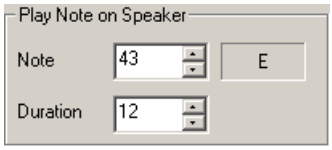
Serial Transmit
This action transmits data out one of the terminal's serial ports.


Transmit out Serial Port
Select Serial Port
COM1
Message to Transmit
"Sends tank level out COM1 to your controller "
Pause until transmission completed

From the "Select Serial Port" drop-down list, select the communications port on the OptoTerminal from which the data will be sent. If you have only one, select **[COM1]**.

In the "Message to Transmit" box, enter the string to send out the serial port or a description of the value to be sent. Enable **Pause until transmission completed** only if you want the terminal to pause. Generally, you should leave this disabled as it may decrease performance on the terminal.

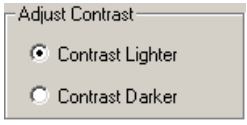
Play Note
This action sounds a tone on the terminal's speaker.



Select which note to play at the "Note" field. Valid values range from 1 (very low pitch) to 86 (very high pitch). The standard beep when a button is pressed is 30. Selecting 0 generates no sound.

At the "Duration" field, select the duration of the note in milliseconds. The standard button beep lasts 100 ms. Refer to the *OptoTerminal Programmer's Reference Manual* section 4.8.1, "PlayNote" for more information.

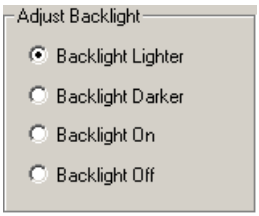
Set Contrast
This action changes the level of contrast on the terminal screen when the event occurs.



For example, you might use this action to provide a contrast adjustment in case the display becomes difficult to read. Create two buttons, one for lighter contrast and one for darker contrast. Each event (e.g., button press) changes the contrast one step lighter or darker.

NOTE: contrast level temporary and TFT displays
The contrast level set with this action is not permanent. When the terminal is reset, it reverts to the default contrast level. If you want to save the contrast level, use a Display-Setting object. Terminals with TFT displays ignore this command as TFT displays do not support a contrast setting.

Set Backlight
This action brightens, darkens, turns off, or turns on the display backlight.



For example, you might use this action to provide a backlight adjustment if the terminal is used in low light situations or in situations in which the light levels may vary dramatically. You may also want to turn the backlight off to extend its life.

Create two buttons for each adjustment: one for lighter backlight and one for darker backlight, one to turn the backlight on and one to turn it off. Each event (e.g., button press) changes the backlight as follows:

- Backlight Lighter and Backlight Darker adjust the backlight one level brighter or darker with each event.
- Backlight Off turns the backlight off.
- Backlight On turns the backlight on to the level it was at before turning it off.

Custom Action
This option allows you to program a custom action if an available Event Builder action does not meet your requirements. Any action supported by the Qlarity programming language may be used. Complex actions and events that require the full expressiveness of the Qlarity programming language, such as loops and if/then/else logic, should be entered in Code View.

Insert a Blank Line
This action is not available from the drop-down list of actions. It appears only if the selected event has been previously edited in Code View and blank lines were inserted in

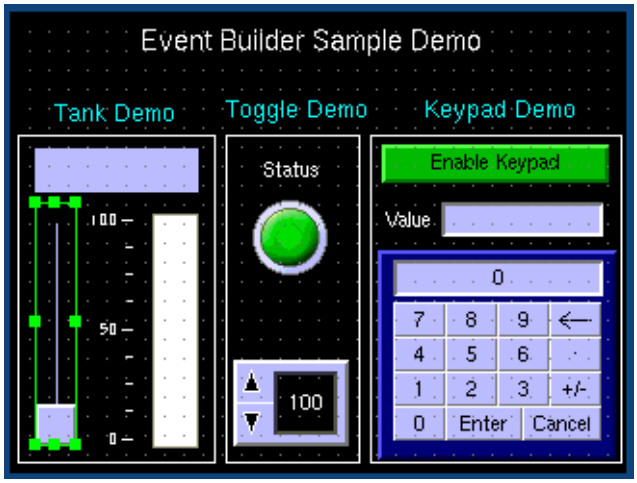
the event's code. This action does nothing. It is used to format the code generated by Event Builder.

8.4.4 Load Event Builder Sample Workspace

A sample workspace, *eventbuilder.qly*, containing events created with Event Builder is included with Qlarity Foundry. You may find it helpful to load the sample workspace and view examples of the objects, events, and actions that can be set up using Event Builder.

To open the sample workspace, click **Open** and go to the folder: *...Qlarity Foundry/Samples/Eventbuilder demo*. Select the file *eventbuilder.qly.*

An image of the workspace is shown below.



The functions in the sample application are described in the following sections.

8.4.4.1 Tank Demo

The Tank Demo controls the level of liquid in a tank with a slider. When the position of the slider in the Tank_Slider object changes, it triggers the following actions:

- Transmits its current value out the serial port to a controller that adds liquid to or drains liquid from a tank to achieve the required level.
- Changes the text label above the slider to the current value of the slider.
- Sets the fill level of the gauge. (In a real-life situation, the controller to which you transmit the value would start the process to fill or drain the tank and send a value back to the gauge to update it. However, in the sample simulation, the value is sent to the gauge directly.)
- Plays a musical note when the slider is moved.

8.4.4.2 Toggle Demo

The Toggle Demo toggles a spinner on and off. When on, a spinner object is displayed that allows a user to increase/decrease the value shown. The Status_toggle object (the button) uses the following events:

TurnOn – triggers the following actions:

- Changes the caption above the toggle button to reflect the current status of "On."
- Enables the spinner object.

TurnOff – triggers the following actions:

- Changes the caption above the toggle button to reflect the current status of "Off."
- Disables the spinner object.

8.4.4.3 Keypad Demo

The Keypad Demo provides a keypad that can be used to enter numerical data to be sent to a controller. The following objects have programmed events:

Enable_button
A press or "click" event enables the numeric keypad (makes it viewable) so it can be used for data input.

Numeric_keypad
Two actions have been programmed to the keypad:

- When the **<Enter>** key is pressed, any data entered (the value) is posted to the Keypad_value text box located above the keypad.
- When the **<Enter>** key is pressed, the value is sent out the serial port to a controller expecting a value.

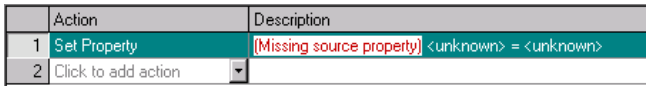
8.4.5 Qlarity Code and Event Builder

While Event Builder requires no programming experience, it uses the Qlarity programming language. Events created with Event Builder generate Qlarity program code. This means that it is possible to develop events in Event Builder and then add advanced functionality in Code View. It is also useful if you are learning the Qlarity programming lan-

guage to create events using Event Builder and then study the program code generated. If you modify an event in Code View and later open the event in Event Builder, your changes are reflected.

8.4.6 Troubleshooting

If you make a mistake while configuring an action, an error message is displayed in red in the "Description" column of the action table.



If any error messages are shown, you will be unable to click **[OK]** to save and exit the dialog box. In some cases, Qlarity Foundry may not detect the error until you click **[OK]**. In those cases, Event Builder remains open for you to correct the error. After you correct the error, click [OK] again. If there are no more errors, Event Builder closes and the event is configured.

If you cannot, or do not want to, correct the error, click **[Cancel]**. Event Builder closes and any changes you made are not saved.

Advanced users may click **[Edit in Code View]** to edit the object template code to correct any errors. **Errors must be corrected in Code View before you can exit or switch back to Layout View**.

8.5 Communication Objects

The Common library (common.lib) provided with Qlarity Foundry contains object templates that are programmed to send data to a computer or other compatible device through a serial or Ethernet port. Once a communication object (or objects) is set up, you can use Event Builder (section 8.4) or EZObjects (Appendix B) to set up an event to send data out the port. No programming is required.

8.5.1 Serial Objects

To use the serial port to send data, add a serial object to the workspace. The optional second serial port can also be used by adding a second serial object.

Following are the properties of a Serial object.

(Name)
Type a name for the new object (type over the default name). Each object in the workspace must have a unique name. An object name has no size limitation but must start with a letter. A name cannot contain spaces but may use the underline character (_). The percent (%), pound (#) and dollar sign ($) symbols can be used at the end of the name.

enabled
If the object is to be enabled, select **true** from the drop-down list. To disable the object, select **false**.

parent
To make the object the "child" of a "parent" object, select a container object from the drop-down list, and the object instance is automatically moved and linked to the selected object in the Object Tree. To move an object to the root container, select "default."

comport
From the drop-down list, select the communications port on the OptoTerminal from which the data will be sent. If you have only one, select **COM1**.

sendtoport
This property receives the data string that is to be sent through the serial port. Data can be sent to this property from a serial object configured in Event Builder (see section 8.4) or an EZObject (see Appendix B). When the serial object's "sendtoport" property receives data, it automatically sends the data followed by a semicolon (;) out the port.

appendterminator
This property determines whether a semicolon is appended to the end of all data sent out the serial port.

8.5.2 Ethernet Objects

To use the Ethernet port to send data, add one or more Ethernet objects. If you want to send data to multiple addresses on a network, you can create a separate Ethernet object for each address; or you can send to different addresses from a single Ethernet object by changing the network address in the object instance as required.

Following are the properties of an Ethernet object.

(Name)
Type a name for the new object (type over the default name). Each object in the workspace must have a unique

name. An object name has no size limitation but must start with a letter. A name cannot contain spaces but may use the underline character (_). The percent (%), pound (#) and dollar sign ($) symbols can be used at the end of the name.

enabled
If the object is to be enabled, select **true** from the drop-down list. To disable the object, select **false**.

parent
To make the object the "child" of a "parent" object, select a container object from the drop-down list, and the object instance is automatically moved and linked to the selected object in the Object Tree. To move an object to the root container, select "default."

sendtoport
This property receives the data string that is to be sent through the Ethernet port. Data can be sent to this property from an Ethernet object configured in Event Builder (see section 8.4) or an EZObject (see Appendix B). When the Ethernet object's "sendtoport" property receives data, it automatically sends the data followed by a semicolon (;) out the port.

protocol
Enter the protocol to be used to send the data (e.g., UDP or TCP).

localport
Enter the address for the UDP or TCP port on the Qlarity-based. For additional information, refer to your Ethernet protocol or contact your network administrator.

foreignport
Enter the address for the UDP or TCP port on the host. For additional information, refer to your Ethernet protocol or contact your network administrator.

targetipaddress
Enter the IP address of the device to which you are sending data.

connected
This property reflects the connected status of the Ethernet object.

appendterminator
This property determines whether a semicolon is appended to the end of all data sent out the Ethernet port.

8.5.3 Receive Data

You can send data from an external device (e.g., computer) to a OptoTerminal via the port (serial or Ethernet). To do this, you specify an object and its property to receive the data in a user application. This changes the specified property value to the received data value.

Use the following format to send data to a user application.

```
<Object name>.<property name>=<value>;
```

(Do not include the angle brackets.)

<Object name>
The name of the targeted object instance.

<property name>
The name of the targeted property (must be entered exactly as it appears in Object Properties).

<value>
The value that you want to insert in the targeted property.

You can also request that an object's property value be sent back to the external device from a OptoTerminal by sending the object name and property as follows.

```
<object name>.<property name>;
```

(Do not include the angle brackets.)

<Object name>
The name of the targeted object instance.

<property name>
The name of the targeted property (must be entered exactly as it appears in Object Properties).

The value of the property followed by a semicolon (;) is sent back out the terminal's port to the device making the query.

NOTE: property value requests
You cannot request property values for the following properties: color, bitmap, ttfont, and bdffont.

8.6 Test the User Application

You can test most of your application's functionality in Simulation mode before you download it to the terminal. If the application uses serial communication, you may want to

configure the Simulation mode to use your PC's serial port(s) to simulate serial communication (refer to section 6.5.1, "Serial Port Setup").

8.7 Save and Compile a Workspace

8.7.1 Save a Workspace

You should save your workspace often while you are working on it to prevent losing work in the event of a computer lockup or power failure. To save a workspace, click on the toolbar or select **Save Workspace** from the File menu. For more information on saving a workspace, refer to section 4.4.

8.7.2 Compile a Workspace

You should compile your workspace periodically while you are working on it to properly display the changes in Layout View. Compiling is also required before you download a user application. Click on the toolbar, or select **Compile Workspace** from the File menu to compile the current workspace into the format required for a user application.

If the compile is successful, the program displays the workspace in Layout View. If any errors occur during the compile, the program switches to Code View, and the error messages are displayed in the Compile dialog box.

8.8 Download a User Application

Before you can download a user application to the OptoTerminal, you need to configure communications in Qlarity Foundry and at the terminal and prepare the terminal for download.

For information on setting up communications, refer to section 7.1.

For information on preparing the terminal for downloading, refer to section 7.2.1.

For information on downloading a user application to the terminal, refer to section 7.2.2.

CHAPTER 9

INTERMEDIATE DESIGN

Chapter 8, "Basic Design" introduces some basic application design techniques using Qlarity Foundry. It also provides information on Qlarity Foundry's Event Builder, which can be used to add events to your applications with no programming. While this is enough for many applications, if you require very complex, flexible, or custom applications, you will want to have some Qlarity programming skills.

To assist you with intermediate design and help you get started with Qlarity Programming, this chapter provides information on the Programmable Objects library (progobjs.lib) provided with Qlarity Foundry. With these objects, you use "method overrides" to manipulate the object templates. This chapter also discusses adding global code and how to create new object templates based on templates that already exist in libraries. By applying a small amount of programming, you can add a great deal of flexibility and power to your applications.

Before proceeding, you should be familiar with the concepts presented in Chapter 8, "Basic Design." Also, this chapter includes many references to the *OptoTerminal Programmer's Reference Manual*. This manual provides an invaluable resource as you learn intermediate Qlarity design and eventually, advanced Qlarity design as explained in Chapter 10.

This chapter covers the following information:

Viewing the Code

Understanding Qlarity for Intermediate Design

- Qlarity Programming Language
- Objects and Templates

Qlarity Code for Objects

- Property Initializations
- Method Overrides

Programmable Objects Library

- Override an Object Method

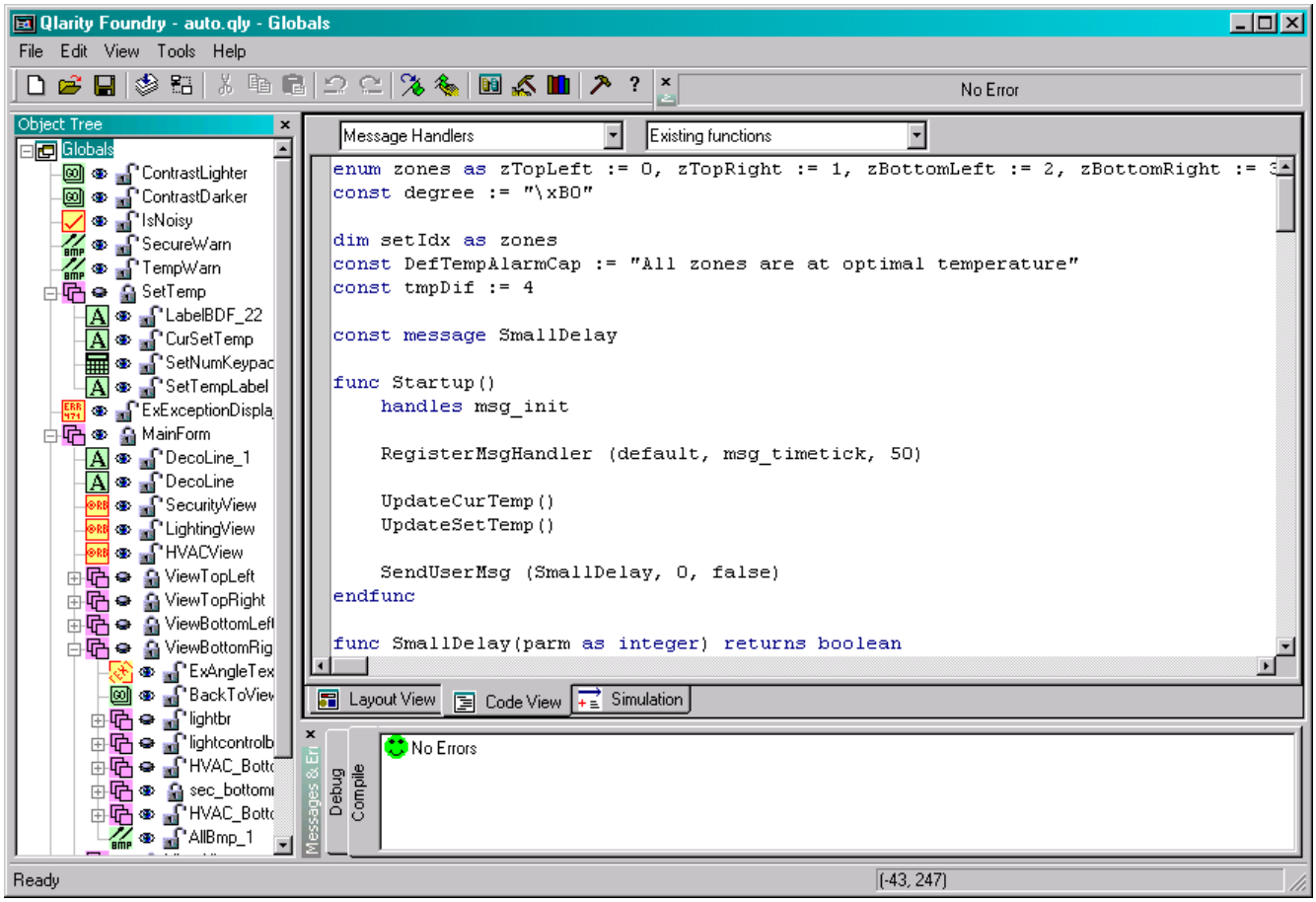
Global Code

- Add a Global Variable to a Workspace
- Add a Global Function to a Workspace
- Add a Global Event Handler to a Workspace

Create a New Object Template

9.1 Viewing the Code

To view and change the Qlarity code for an object, select the object in Layout View or in the Object Tree and then click the Code View tab at the bottom of the work area (see section 3.8, "Layout and Code Views"). In the illustration on the next page, the "global" code for an example workspace is shown. The Messages and Errors window at the bottom shows any compiling errors that occurred during the last compile.



9.2 Understanding Qlarity for Intermediate Design

Before you can effectively design a user application at the intermediate level, you need to understand some basic Qlarity concepts, which are explained in the following sections.

9.2.1 Qlarity Programming Language

Qlarity syntax is based on the BASIC programming language with extensions added to handle creation and manipulation of objects. This chapter provides examples of programming code written in the Qlarity language. For additional examples and information, refer to the *OptoTerminal Programmer's Reference Manual*.

9.2.2 Objects and Templates

A Qlarity application consists primarily of objects. An object template defines a new type of object with certain properties and behaviors. Any number of object "instances" that are based on the object template may be added to an application.

Object templates typically reside in libraries where they can be used in many different applications. However, templates can also exist in a normal workspace. When templates are part of a workspace, they appear at the bottom of the Object Tree when Code View is selected.

A template defines the properties (variables) and methods (functions) for a certain type of object. The properties and methods in a template completely define the behavior of object instances that are created from the template. Properties may also be assigned default values in the template. These values are used if the object instance does not assign an initial value to a property. Details on creating templates are provided in Chapter 10, "Advanced Design."

Each object instance may be customized by changing the values of the properties and by overriding methods defined in the template. A "method override" is a function defined

in the object instance that replaces the method of the same name defined in the template (for the specific instance only). If the functionality of the template method is desired, it can still be called from inside the override method.

Customizing object instances by changing their initial properties requires no programming and is discussed in Chapter 8. Conversely, overriding methods does require some programming and is the main focus of this chapter.

9.3 Qlarity Code for Objects

The following sections describe the Qlarity code used to define an object and the method overrides in an object template.

9.3.1 Property Initializations

Code for a typical object includes several lines in the following format:

```
init <propertyname> := <value>
```

<propertyname>
This is the name of one of the object's properties.

<value>
This assigns the initial value for the property in the application. If a property is not shown, it is assigned the default initial value from the object template. Changing "<value>" in one of these statements is the same as changing the property value in Object Properties.

9.3.2 Method Overrides

Most methods defined in an object template may be overridden in any instance created from the template. Indeed, many objects have methods that are intended only for override; the default method in the template does nothing. This allows an object to provide or "publish" a way to customize the behavior of the object without modifying the template. For example, a timer object might provide a function called Alarm(), which is called by the object each time a preset time period expires. The function in the template does nothing, but it allows instances to override the function to provide a desired response to the timer expiration.

The code for the method override is defined inside the object instance. The override method must have the same name, parameters, and return type as the template method that is being overridden. For example, if the method in the template is named "foo," and it takes an integer and a float as parameters and returns a float, then the override function must also be named "foo" and must also take an integer and a float as parameters (in the same order as the template method) and return a float.

In addition, if the template method handles one or more messages, the override must handle the same message(s). This may require you to examine the code in the template (possibly in a library). Otherwise, the method is like any other Qlarity function and may contain whatever Qlarity code is desired.

The special function, Default(), when placed inside an override method, calls the overridden method in the template. The same parameters that were passed to the override method should be passed in Default().

Refer to the *OptoTerminal Programmer's Reference Manual*, section 2.12, "Functions" for information on how functions are defined in Qlarity.

NOTE: create a new template
If you find that you are adding identical override functions to many instances of the same object, consider modifying the object template to create a new template with the desired functionality.

9.4 Programmable Objects Library

The Programmable Objects library (progobjs.lib) contains object templates that are similar to the templates in the EZObjects library. However, these objects are intended to handle events through method overrides rather than through "event" property settings. Almost every Programmable Object has one or more methods that can be overridden in object instances to provide specific object behavior.

For example, many button-like objects (such as the ButtonBDF or LabelBDF objects) publish a function called Click(). By default, this function does nothing, but each instance of this object can include an override of the Click() function to take some action in response to a press event on the object. A typical response might include sending some characters to a communications port, switching to a different screen, or modifying the contents of another object.

In this manner, an object's response to events is programmed using the Qlarity programming language. Once you are familiar with the basics of Qlarity programming, it

is easy to program complex event handling into objects without the need to manage numerous "event" properties.

9.4.1 Override an Object Method

To override an object method in your application, do the following:

1. Add the Programmable Object that will be used to handle the event (e.g., ButtonBDF, Timer).

2. Select the object in the workspace in Layout View or in the Object Tree, and select **Code View**.

3. From the **[Events/Overrides]** drop-down list above the code, select the method you want to override.

   Select **[Show only event functions]** if you want the drop-down list to displays only methods that have been designated "event methods." If **[Show only event functions]** is not selected, all possible override methods are listed. Move the mouse pointer over a method in the list to display a description of the function.

4. When you select a method from the list, code similar to the following appears beneath all code for the object:

   ```
   func click()

   endfunc
   ```

5. Between the "func" statement and the "endfunc" statement, add Qlarity code to implement the desired response to the event.

   A typical event response might include code to set properties in this or another object. The Qlarity code to do this is as follows:

   ```
   <objname>.<propertyname>=<value>
   ```

   <objname>
   This is the name of the object whose property will be changed. To assign values to its own properties, the override function in an object can refer to itself using the "me" keyword as the "<objname>."

   <propertyname>
   This is the name of the property to change.

   <value>
   This is the new value to assign to the property.

   Another common activity is switching screens (display pages) or hiding one object and displaying another. To switch screens, the method typically disables the current screen object and enables the screen object for the desired new screen, as follows:

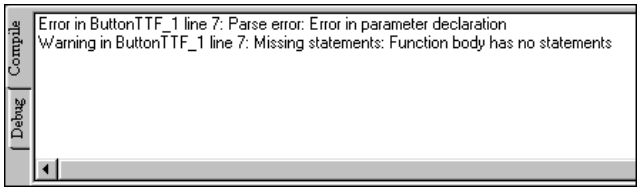
   ```
   curscreen.enabled = false
   newscreen.enabled = true
   ```

   where "curscreen" and "newscreen" are the names of the screen objects to be manipulated.

   Since the override method is a complete Qlarity function, local variables can be declared and code including conditional statements (if-else), looping (for-next, do-while, etc.), and any other legal Qlarity code can be added to the function. Complete details on the Qlarity programming language are found in the *OptoTerminal Programmer's Reference Manual*.

NOTE: return from functions
Since Qlarity is an event-driven program, you should return from functions in a timely manner. Writing code that does not return will cause the system to hang.

6. Click (Compile button), or select Compile from the File menu to compile the workspace. If there are compile errors in your code, messages are displayed in the Messages and Errors window. An example of such a message is shown below.


Compile
Debug
Error in ButtonTTF_1 line 7: Parse error: Error in parameter declaration
Warning in ButtonTTF_1 line 7: Missing statements: Function body has no statements

   The message includes a description of the error, including the object and line number where the error occurred. Double-clicking the error message in the Messages and Errors window takes you to the location of the error in Code View.

NOTE: no errors
If there are no errors found during the compile, the workspace is automatically returned to Layout View.

9.5 Global Code

In addition to method overrides, an application can be customized by adding variables and functions to the Global Code space. Global variables can be used to hold data not stored in objects. Global functions can implement commonly performed routines that might be shared by many objects. Also, most messages can be handled by global "event handlers" if desired. Event handlers are special functions that are called by the system software when the message that the function "handles" is processed.

Since global functions are not overrides, there are few restrictions on the names, parameters, or return type for global functions.

9.5.1 Add a Global Variable to a Workspace

A global variable is a variable that is declared in the Global Code space, outside of any object. To add a global variable to the workspace, do the following:

1. Select the Global Code space by clicking **Globals** in the Object Tree.

2. Click the **Code View** tab. The editor window displays any current global source code.

3. On a blank line in the editor window (separate from any other function), declare a new variable by typing the "dim" (dimension) keyword, followed by the name of the variable, the keyword "as," and then the type of the new variable.

   For example, the following code declares a new integer variable named "count":

   ```
   dim count as integer
   ```

4. If desired, also assign an initial value to the variable by adding a new line containing the keyword "init," the name of the new variable, the ":=" operator, and the initial value of the variable.

   For example, the following line initializes the new count variable to a value of 10:

   ```
   init count := 10
   ```

5. Repeat steps 3 and 4 for as many variables as desired.

6. Click or select **Compile** from the File menu to compile the workspace. Any errors in the new source code are identified by messages in the Messages and Errors window. If no errors are present, the compiled workspace is returned to Layout View.

9.5.2 Add a Global Function to a Workspace

To add a global function to the workspace, do the following:

1. Select **Globals** in the Object Tree to access the Global Code space.

2. Select **Code View**. The editor window displays any current global source code.

3. On a blank line in the editor window, outside of any other function, declare a new function by typing the "func" keyword, followed by the name of the function, a left parenthesis, the parameters of the new function, and a right parenthesis. If the function returns a variable, the right parenthesis should be followed by the keyword "returns," then the type of the returned value. The end of the new function is indicated by the "endfunc" keyword (on its own line). The parameters appear as new variable declarations (see section 9.5.1) without the "dim" keyword. Each parameter is separated from other parameters by a comma.

   For example, the following code declares a new function named "average" that takes two floats (named first and second) as parameters and returns a float.

   ```
   func average(first as float, second as ->
   float) returns float
   endfunc
   ```

NOTE: code formatting
A statement is defined as a single line of code. In order for the compiler to distinguish between statements, each statement must be separated by a "newline" character. If a statement must be split into multiple lines, type `->` at the end of the line to tell the compiler to look for the rest of the statement on the next line.

4. Fill the body of the function (between the "func" statement and the "endfunc" statement) with the Qlarity code to implement the desired function. If the function returns a value, be sure to include a return statement, followed by an expression that yields the value being returned. For more information, see the *OptoTerminal Programmer's Reference Manual*.

In the example above, the completed function is as follows:

```
func average(first as float, second as ->
float) returns float
    return (first + second)/2.0
endfunc
```

5. Repeat steps 3 and 4 for as many functions as desired.

6. Press or select **Compile** from the File menu to compile the workspace. Any errors in the new source code are identified by messages in the Messages and Errors window. If no errors are present, the compiled workspace is returned to Layout View.

9.5.3 Add a Global Event Handler to a Workspace

Event handlers are functions that are called by the system in response to a message generated by an event, such as a touch screen press or the receipt of characters through a serial port. To add a global event handler to the workspace, do the following:

1. Select the Global Code space by clicking **Globals** in the Object Tree.

2. Select **Code View**. The editor window displays any current global source code.

3. On a blank line in the editor window (outside of any other function), declare a new function by typing the "func" keyword, followed by the name of the function, a left parenthesis, the parameters of the new function, and a right parenthesis. If the function returns a value, the right parenthesis should be followed by the keyword "returns" and then the type of the returned value. Indicate the end of the function with the "endfunc" keyword (on its own line). The parameters appear as new variable declarations (see section 9.5.1) without the "dim" keyword. Each parameter is separated from other parameters by a comma.

   Since this is a message handler, there are restrictions on the parameters and return type for this function. The allowed parameters and return type depend on what message is being handled. There are no restrictions on the name of the function except those imposed by Qlarity. (See the *OptoTerminal Programmer's Reference Manual*, Chapter 3, "Messages and Message Handler Prototypes" for more details.)

   Also, a "handles" statement is needed on the line following the function declaration. The statement begins with the keyword "handles" followed by the name of the message that is being handled by the function.

   For example, the following code declares a new event handler named "timer" that handles MSG_TIMETICK. As explained in the *OptoTerminal Programmer's Reference Manual*, Chapter 3, the handler takes no parameters and does not return a value.

```
func timer()
    handles MSG_TIMETICK
endfunc
```

4. Fill the body of the handler (on lines between the "handles" statement and the "endfunc" statement) with the Qlarity code to implement the desired function. If the handler returns a value, be sure to include a "return" statement. For more information, see the *OptoTerminal Programmer's Reference Manual*.

   For example, the handler described above could check the value of a global variable named "threshold" and enable an object named "cal_screen" if threshold is greater than 10. The complete function example follows:

```
func timer()
    handles MSG_TIMETICK

    if (threshold > 10) then
        cal_screen.enabled = TRUE
    endif
endfunc
```

5. Repeat steps 3 and 4 for as many handlers as desired. Do not add two handlers for the same message.

6. Click or select **Compile** from the File menu to compile the workspace. Any errors in the new source code are identified by messages in the Messages and Errors window. If no errors are present, the compiled workspace is returned to Layout View.

9.6 Create a New Object Template

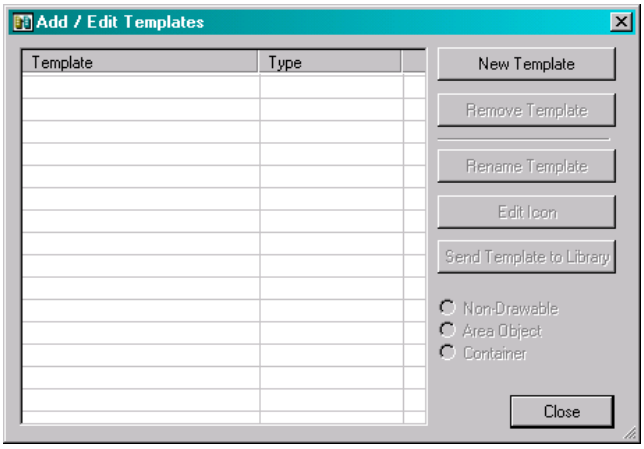
The method override mechanism provides a powerful way to customize an individual object's behavior and response to events. However, you may want to change the behavior of all instances of a particular object. Adding an identical override to all objects from a particular template would be

tedious and prone to error. It is better to modify an existing object template to create a new object template.

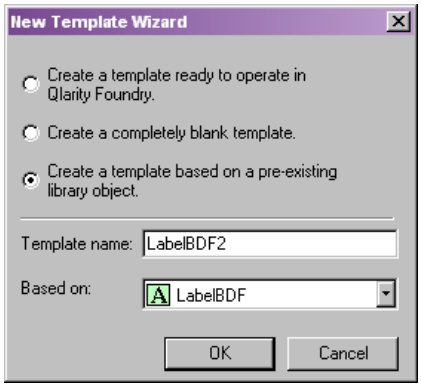
NOTE: do not modify code in libraries
It is recommended that you do not directly modify the code in the Opto 22 standard object libraries. Opto 22 occasionally updates these libraries, and your changes would be lost. Rather, you should copy the template code into your workspace or a personal library (e.g., mylib) and make your modifications there.

Do the following to create a new object template based on a library object template:

1. Click on the toolbar, or select **Add/Edit Templates** from the Edit menu, and the Add/Edit Templates dialog box is displayed. (You can also right-click anywhere in the Templates branch of the Object Tree, and click **Add/Edit Templates** on the shortcut menu).



2. Click **[New Template]** and the following dialog box is displayed.



3. Select "Create template based on a pre-existing library object."

   This option creates a new object template using the programming code from a library object you specify in the "Based on" box.

4. From the "Based on" list box, select the existing library object on which you want to base the new object template.

5. In the "Template name" box, type a name for the new template. Each template in the workspace must have a unique name. A template name has no size limitation but must start with a letter or underline character (_). A name cannot contain spaces but may contain the underline character or pound sign (#). The percent (%) and dollar sign ($) symbols can be used at the end of the name.

6. Click **[OK]** to create the new template.

   Many library templates depend on global code or other resources in the library. Qlarity Foundry will add these items to your workspace and inform you of the additional changes in a dialog box. When the dialog box appears, click **[OK]** to proceed.

7. The template list in the Add/Edit Templates dialog should now include the new template. Click **Close** to close the dialog box.

Now do the following to edit the new template to your specifications.

1. Select **Code View**. The new template is now listed under the Templates heading in the Object Tree.

2. Select the new template in the Object Tree. The source code for the template is displayed in the editor window.

3. Edit the template source code. Variables declared inside a template become properties of the template object, and functions declared inside the template become the object's methods. Properties and methods can be added, deleted, or modified as desired. You can add properties and methods just as described for the Global Code space (see section 9.5).

4. When you are finished editing the template, click or select **Compile** from the File menu to compile the

workspace. Any 5s in the new source code are identified by messages in the Messages and Errors window. If no errors are present, the compiled workspace is returned to Layout View.

Notice that the Object Palette contains a new icon representing the new template. By default, the new template's icon is the same as the icon for the library template that served as the basis for the new template (except that the background color is different). Icons for local templates in a workspace appear before library template icons.

5. To edit the icon or change the name, click on the toolbar, or select **Add/Edit Templates** from the Edit menu. Click the template name in the Add/Edit Templates dialog box, then click **[Edit Icon]**. Refer to section 5.1.2, "Edit Icon" for more information.

9.7 Where to Go From Here

This chapter introduced the basics of programming applications using Code View and the Qlarity programming language. As your familiarity with Qlarity increases, your ability to program complex tasks and customize your application will grow. To strengthen your background in Qlarity, you should read the *OptoTerminal Programmer's Reference Manual*, in particular Chapter 1, "QTERM-G70/G75 Software Fundamentals" and Chapter 2, "Qlarity Language Syntax".

Next, proceed to Chapter 10, "Advanced Design" in this manual to help you get started producing your own object templates.

CHAPTER 10

ADVANCED DESIGN

The ability to completely design custom objects for a user application is the crowning feature of the Qlarity programming language. This chapter provides information to help users design their own objects using Qlarity.

Advanced users are not bound by the selection of objects available in Qlarity and Opto 22 libraries. As an advanced user, you can design objects that best represent the data and user interface elements for a particular application. For example, if the OptoTerminal is replacing an existing control panel, you can create objects that mimic the "look and feel" of the user interface elements on the old panel. The variety of objects you can create with Qlarity is virtually unlimited.

This chapter introduces the Qlarity concepts that are crucial for object design, including the following:

- Validation Functions
- The Qlarity API Library
- Exception Handling

It also includes information on the following:

- Create a New Object Template
- Guidelines for Designing New Object Templates

Before proceeding, you should be familiar with the concepts presented in Chapter 8, "Basic Design" and Chapter 9, "Intermediate Design" and have some practice with Qlarity design at those levels. In addition, you should review Chapter 1, "QTERM-G70/G75 Software Fundamentals" and Chapter 2, "Qlarity Language Syntax" of the *OptoTerminal Programmer's Reference Manual*, as these cover basic information needed for programming in Qlarity.

10.1 Validation Functions

Validation functions are special functions that are closely associated with object properties or global variables. When a property is assigned a value using the validation assignment operator (=), the associated validation function is implicitly called and passed the new value as a parameter. This facility allows objects to appropriately respond to property changes. Validation functions are also called when certain API functions (such as Val() and SetObjProp()) are used to assign a value to a property.

Validation functions are also useful for assuring that the new value is valid or within an appropriate range (hence the term "validation function").

To associate a validation function with a property, the function must have the same name as the property. In almost all cases, it must take a single parameter of the same data type as the property. The name of this parameter can be any legal Qlarity variable name, but it is good style to use a common name for all your validation functions (see section 10.5, "Guidelines for Designing New Object Templates"). Validation functions should not have a return value. Apart from these restrictions, the function is written and behaves as a normal function. The validation function can be explicitly called from other Qlarity functions if desired.

For example, when the graph level property of a bar graph object is changed, you need to redraw the graph at the new level. Also, values outside the range of the gauge should be "clipped" to a maximum or minimum value. These can both be accomplished with a validation function for the graph level property. Suppose that the property is an integer named "graphlevel." The validation function might look like the following:

```
func graphlevel(newval as integer)
    'clip to the max or min value
    if (newval > maxval) then
       newval = maxval
    elseif (newval < minval) then
       newval = minval
    endif
    'Now assign newval to level
    'Note the use of strict assignment op.
    graphlevel := newval
    'Now request this object to redraw
    rerender(me)
endfunc
```

The Rerender() function is a Qlarity API function (see section 10.2, "The Qlarity API Library") that generates a MSG_DRAW message for the area of the object.

For array properties, separate validation functions can be written for assignment of the entire array and assignment of individual elements of the array. See section 2.12.4, "Array Validation Functions" and section 2.12.5, "Array Element Validation Functions" of the *OptoTerminal Programmer's Reference Manual* for more detail.

Validation functions may also be associated with global variables. These global validation functions should be included in the Global Code space.

NOTE: strict assignment operator
Ordinarily, the validation function assigns the new value to the property using the strict assignment operator (:=). It is important to use the strict assignment operator when assigning the new value to the property inside the validation function. If the validation assignment operator is used, the validation function recursively calls itself until the system software stack overflows.

Refer to section 2.12.3, "Validation Methods" in the *OptoTerminal Programmer's Reference Manual* for additional details on validation functions.

10.2 The Qlarity API Library

The Qlarity API (Application Programming Interface) is a library of functions built into the Qlarity system software. These functions allow Qlarity applications to interact with the Qlarity-based hardware to perform tasks such as drawing on the display or sending data to a communications interface. Other functions perform common tasks that would be tedious or difficult to implement using the Qlarity language. Over 100 functions are available. All functions are documented in Chapter 4, "Qlarity API Function Reference" of the *OptoTerminal Programmer's Reference Manual*.

10.3 Exception Handling

When an abnormal condition arises while a Qlarity application is running, the system software generates an exception. An application may also throw an exception when it detects a problem. The Qlarity exception handling system allows the exception handling code to be localized to a convenient location, either within an object template or for the application as a whole. This section describes the facilities that Qlarity provides for generating and handling exceptions.

An exception is generated in one of two ways: the system software throws an exception if it detects a problem, or the application can throw an exception for any reason. The application exception is thrown with a call to the Throw() API function.

Each exception carries certain information regarding what occurred and where. This information includes an exception level (an indicator of severity), an exception type, a location where the exception occurred, and a brief explanation of the error.

The exception level can assume the following values (in order of decreasing severity):

EXLEV_COMPILER
This level indicates a serious error most likely caused by the Qlarity compiler.

EXLEV_SYSTEM
This level indicates a problem in the system software (such as memory exhaustion, a problem with the message queue, and so on).

EXLEV_IGNORABLE
This level indicates an unexpected condition that is generally serious but not fatal.

EXLEV_USER
This is the exception level for exceptions thrown by the application (via the Throw() API function).

EXLEV_MIN
This level represents the minimum severity for an exception.

The exception type is an error code that describes the cause of the error. Each exception type is accompanied by a short text description of the problem. The error code provides a convenient means for Qlarity exception handlers to determine what error occurred, while the description provides a readable summary of the exception. The location string indicates the Qlarity function or system software facility that was executing when the error occurred. This is useful for debugging the application.

The error codes and descriptions are listed in Appendix B, "Exception List" of the *OptoTerminal Programmer's Reference Manual*.

Exception handling code is supported in the Qlarity programming language by "check error"/"on error" blocks. When an exception occurs, the Qlarity execution engine determines if the offending code is inside a "check error" block. If so, execution immediately jumps to the first statement in the "on error" block. If the offending code is not in a "check error" block, the Qlarity execution engine checks the function that called the current Qlarity function for an enclosing "check error" block. This process iterates until either an enclosing "check error" block is found, or the message handler for the current message is reached. Since the message handler was called by the system in response to an event, the call cannot be traced further than this function.

For example, suppose a Qlarity statement in function, Foo(), causes an exception. Foo() is in turn called by a function named Bar(). Bar() is a handler called by the system in response to a message. When the exception occurs, the Qlarity execution engine first checks the function, Foo(), to see if the offending statement is enclosed within a "check error" block. If it is, execution jumps to the first statement in the "on error" block that follows the "check error" block. If it is not enclosed in a "check error" block, the Qlarity execution engine then checks the Bar() function to determine if the call to Foo() was enclosed in a "check error" block. If it is, execution jumps immediately from the offending statement in Foo() to the first statement in the corresponding "on error" block in Bar(). If no enclosing block is found in Bar(), the system handles the exception as described below.

Unhandled exceptions are maintained in a LIFO (last in first out) stack by the system software. If multiple unhandled exceptions are pending, a call to GetException() always returns the most recently thrown exception on the stack.

If no enclosing "check error" block is found, an exception remains in the system exception stack until the current message completes processing. The handler that caused the exception is aborted, but the current message continues to propagate to other objects until it has been processed to completion.

At this point, if there are any exceptions in the system exception stack, the system software generates a MSG_ERROR message, which is processed immediately (i.e., before any other pending messages are processed). MSG_ERROR is a special message that can only be processed by handlers in the Global Code space. This allows applications to provide a "last chance" global error handler routine that receives all unhandled exceptions. The MSG_ERROR handler has no parameters, so the exceptions must still be retrieved by calls to GetException().

Retrieving an exception by calling GetException() removes the exception from the system exception stack. Therefore, calling GetException() terminates an exception unless it is rethrown by calling the Rethrow() API function.

If the exception has not been handled (i.e., removed from the system exception stack) after MSG_ERROR is processed, or if there is no global MSG_ERROR handler, the exception is either discarded (for exception levels of EXLEV_IGNORABLE or lower), or the exception location and description are transmitted from the primary serial port (for exceptions of EXLEV_SYSTEM or higher).

The typical "check error" block or global MSG_ERROR handler calls GetException() to retrieve information for the last thrown exception, check the error type, and respond appropriately to each type of anticipated exception. If the error type is not what was anticipated, the exception should be rethrown using the Rethrow() API function. This gives higher level exception handling code the opportunity to handle the exception if desired.

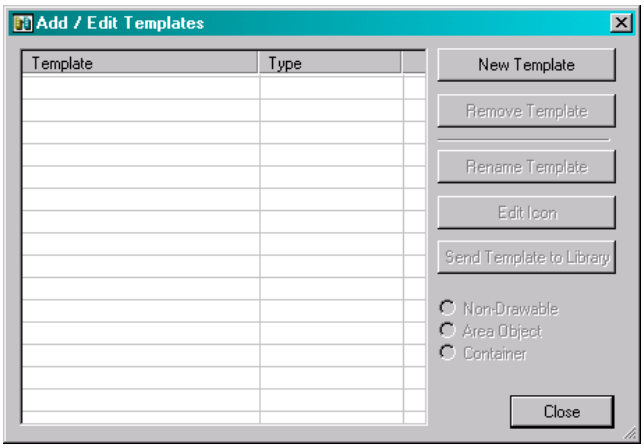
NOTE: GetException()
Ordinarily, GetException() should be called only ONCE in an "on error" block. In a global MSG_ERROR handler, GetException() is typically called repeatedly until all exceptions have been retrieved.

Details on the syntax of "check error"/"on error" can be found in section 2.15, "Exception Handling" in the *OptoTerminal Programmer's Reference Manual*.

10.4 Create a New Object Template

To create a new object template in your application, do the following:

1. Click on the toolbar, or select **Add/Edit Templates** from the Edit menu. The Add/Edit Templates dialog box is displayed.



(You can also right-click anywhere in the Templates branch of the Object Tree, and click **Add/Edit Templates** on the shortcut menu).

2. Click **[New Template]** and the following dialog box is displayed.

3. Select one of the following:

Create a template ready to operate in QF
This option inserts the new template boilerplate code into your workspace (see section 10.4.1, "New Template Boilerplate Code").

Create a completely blank template
This option starts a blank template. You will need to add all Qlarity code.

4. Type a name for the new template in the "Template Name" edit box.

5. Select the type of object (non-drawable, area object, or container) from the "Object Type" list box.

6. Click **[OK]**. The new template name appears in the Add/Edit Templates dialog box.

7. Click **[Close]** to close the Add/Edit Templates dialog box.

8. Select **Code View**. The new template name appears under the Templates icon in the Object Tree. Templates are not shown in the Object Tree in Layout View.

9. Click on the new template in the Object Tree. If you created a template "ready to operate in QF," the boilerplate code appears in the edit window.

10. Variables declared inside a template become properties of the template object, and functions declared inside the template become the object's methods. Add, delete, or modify properties and methods as desired.

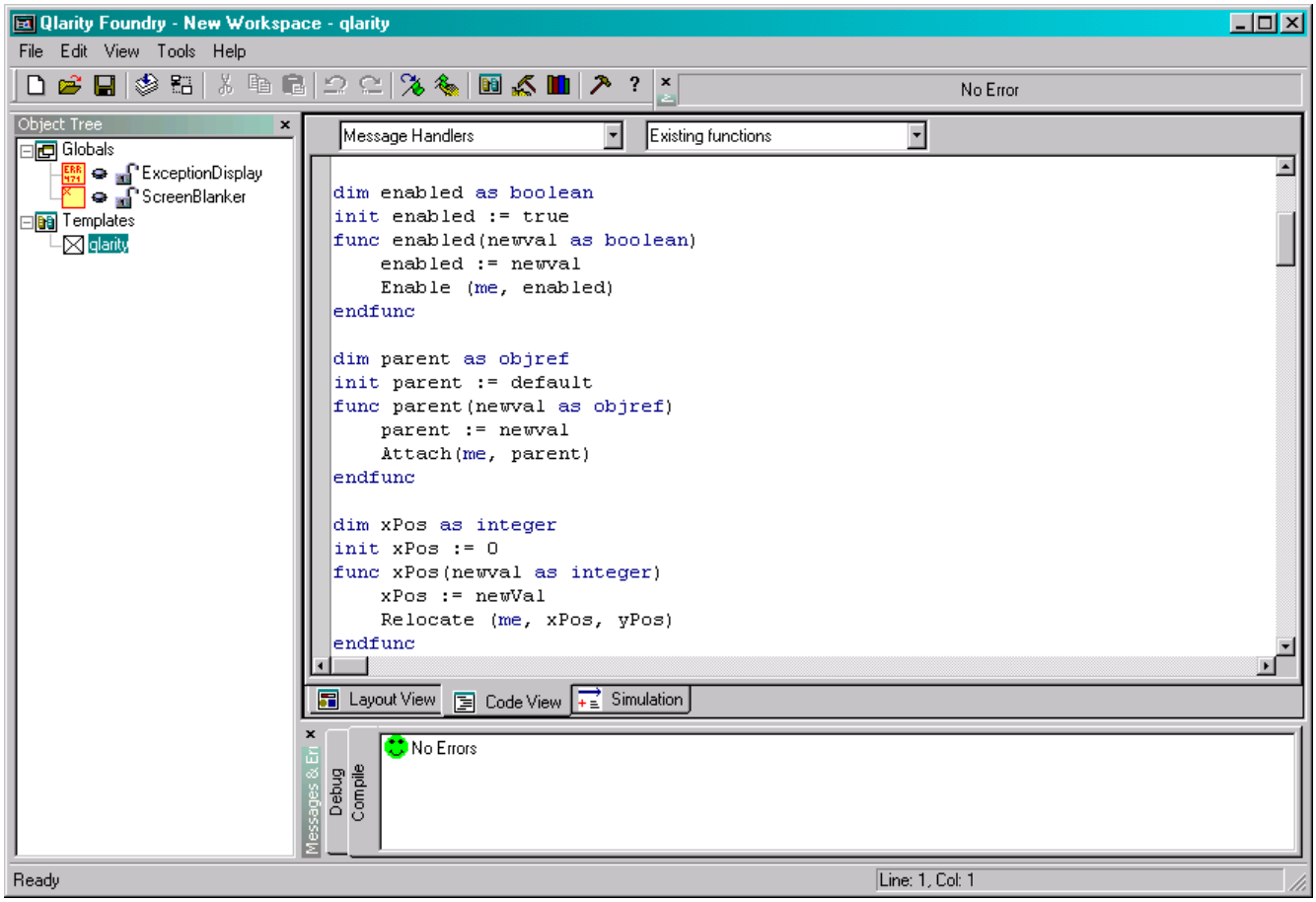
11. When you are finished defining the template, click or select **Compile** from the File menu to compile the workspace. Any errors in the new source code are identified by messages in the Error Messages box. If no errors are present, the compiled workspace is returned to Layout View.

    Notice that the Object Palette contains a default icon representing the new template.

12. To edit the icon or change the name, click on the toolbar, or select **Add/Edit Templates** from the Edit menu. Click the template name in the Add/Edit Templates dialog box, then click **[Edit Icon]**. Refer to section 5.1.2, "Edit Icon" for more information.

10.4.1 New Template Boilerplate Code

The new template boilerplate code can be used as a foundation for building new objects. Using the boilerplate code can greatly simplify the creation of new objects. This section provides an overview of what the boilerplate code is and the functionality it provides. The actual boilerplate code generated by Qlarity Foundry is continually being improved and may differ from the code described in this section. The functionality, however, is mostly the same.



The illustration above shows a portion of an example of the "boilerplate" code provided by Opto 22 as an optional starting point when creating a new object template.

The boilerplate code included in a new template depends on the type of object being created (i.e., non-drawable, area object, or container). The purpose of the boilerplate code is to provide a set of properties and methods found in a typical object of the selected type. It also provides fully functional tool methods. The source code is well documented and serves as an example of a working Qlarity object.

Each property is defined as follows:

descriptive comment	(describe the property)
dim statement	(create the property)
init statement	(assign a default value)
validation method	(provide functionality for property changes)

For example, the "enabled" property is common to all objects and appears in the boilerplate code for any type of template. The code that defines the property is as follows:

```
'Whether the object is displayed on the ->
'terminal
dim enabled as boolean
init enabled := true
func enabled(newval as boolean)
   enabled := newval
   Enable (me, enabled)
endfunc
```

The comment describes the purpose of the property. The "dim" statement creates a boolean property named "enabled." The "init" statement assigns a default value of TRUE to the enabled property, which means that if the property is not assigned an initial value in the object instance, the value defaults to TRUE.

Finally, the validation function (see section 10.1, "Validation Functions") describes the behavior of the object when

the value of "enabled" is changed. The validation function first does a strict assignment of the new value to the property then calls the Enable() API function to inform the firmware that the enabled status of the object has changed. Calling API functions to implement the effects of a property change is a very common activity in validation methods.

An example of another property found in all versions of the boilerplate code is the "parent" property, as follows:

```
'Who do we attach to
dim parent as objref
init parent := default
func parent(newval as objref)
   parent := newval
   Attach(me, parent)
endfunc
```

This code creates a property of type "objref" named "parent." "Parent" is given a default value of "default," which is the root container when assigned to objref variables. The validation method assigns the new value to the property using the strict assignment operator, then calls the Attach() API function to attach the object to its new parent container.

10.4.1.1 Non-Drawable Objects

The boilerplate code for a non-drawable object template includes the "enabled" and "parent" properties. It also includes a handler method that handles the MSG_INIT message. The code is as follows:

```
'Perform basic initialization for the object
'The function will be called once when ->
'the terminal starts up.
'This function may also be called within ->
'Qlarity Foundry whenever a property is ->
'set in the properties window.
func StartUp()
   handles MSG_INIT

   Enable (me, enabled)
   Attach (me, parent)
endfunc
```

The comments describe what the handler is and when it is called. The method is called "StartUp," takes no parameters, and has no return value. It is declared as a handler for the MSG_INIT message.

The body of the method merely calls the Enable() and Attach() API functions, passing the values of the "enabled" and "parent" properties. This is necessary to properly initialize the objects by informing the system software about the state of the object at system startup.

All versions of the boilerplate code include a set of tool message handlers so that they can properly function in Qlarity Foundry. These methods are described in detail in the next section. Other properties and methods can be added to the boilerplate code to implement the desired behavior in the object.

10.4.1.2 Area Objects

The area object boilerplate code implements a simple rectangular, drawable object. It also provides the "enabled" and "parent" properties, as well as the "xPos", "yPos," "width," and "height" properties to represent the position and size of the object on the display. The code for "xPos" is representative of the code for all the new properties, as shown below:

```
'The horizontal position of the object
dim xPos as integer
init xPos := 0
func xPos(newval as integer)
   xPos := newVal
   Relocate (me, xPos, yPos)
endfunc
```

"xPos" is an integer with a default value of 0. The validation method assigns the new value to the property as previously described, then calls the Relocate() API function to inform the system software that the object has moved to a new location on the display. This generates a MSG_DRAW message for the appropriate regions of the display.

The MSG_INIT handler in the area object boilerplate code is similar to its counterpart in the non-drawable code, as follows:

```
'Perform basic initialization for the object
'The function will be called once when ->
'the terminal starts up.
'This function may also be called within ->
'Qlarity Foundry whenever a property is ->
'set in the properties window.
func StartUp()
   handles MSG_INIT

   Relocate (me, xPos, yPos)
   Resize (me, width, height)

   Enable (me, enabled)
   Attach (me, parent)
endfunc
```

Notice that calls to Relocate() and Resize() have been added to the handler. These properly initialize the location and size of the object on the display.

Since the area object is drawable, a MSG_DRAW handler is required to do the actual drawing. The boilerplate version of this method is as follows:

```
'Display the object.
'This function will be called whenever ->
'necessary to show the object.
func Draw()
   handles MSG_DRAW

   SetFGColor (RGB_WHITE)
   SetBGColor (RGB_GRAY)
   DrawBox (xPos, yPos, xPos + width - 1, ->
   yPos + height - 1)
   DrawLine (xPos, yPos, xPos + width ->
   - 1, yPos + height - 1)
   DrawLine (xPos, yPos + height - 1, ->
   xPos + width - 1, yPos)
endfunc
```

NOTE: code formatting
A statement is defined as a single line of code. In order for the compiler to distinguish between statements, each statement must be separated by a "newline" character. If a statement must be split into multiple lines, type `->` at the end of the line to tell the compiler to look for the rest of the statement on the next line.

This function sets the foreground color to white and the background color to gray. It then draws a box using the object's properties to determine the location and size. The two calls to DrawLine() draw an X inside the box. This code is easily modified to provide the desired drawing behavior.

10.4.1.3 Container Objects

The boilerplate code for a container template is very similar to the code for an area object template. The MSG_DRAW handler is slightly different, as follows:

```
'Display the object.
'This function will be called whenever ->
'necessary to show the object.
func Draw(pass as boolean)
   handles MSG_DRAW
   if not pass then
      SetFGColor (RGB_WHITE)
      SetBGColor (RGB_GRAY)
      DrawBox (xPos, yPos, xPos + width ->
      - 1, yPos + height - 1)
      DrawLine (xPos, yPos, xPos + width ->
      - 1, yPos + height - 1)
      DrawLine (xPos, yPos + height - 1, ->
      xPos + width - 1, yPos)
   endif
endfunc
```

Notice that the MSG_DRAW handler for a container takes a boolean parameter. This is because a container receives two opportunities to handle a MSG_DRAW message: once before the message is passed to its children, and once after the children have finished handling the message. The value that the handler receives in the pass parameter indicates whether or not the message has already been passed to the container's children (see section 3.3, "Draw Messages" of the *OptoTerminal Programmer's Reference Manual*).

10.4.2 Getting New Object Templates to Work in Qlarity Foundry

Since Qlarity Foundry does not have knowledge of the implementation of a Qlarity object, the object template must implement handlers for several tool messages to work properly in Qlarity Foundry. These messages are generated by Qlarity Foundry when the user interacts with the object in Layout View. For example, when an object is selected, it should display a series of resize grips to allow the user to resize the object by dragging the mouse. When the resize grips are dragged, the object should respond accordingly.

Requiring each object to implement its own Layout View behavior allows you to extend the functionality of Qlarity Foundry in unique and powerful ways. This power and flexibility comes at the price of some complexity in writing handlers for the tool messages. Fortunately, the boilerplate code is sufficient for most objects.

This section describes the implementation of the tool handlers in the new template boilerplate code. The tool messages are explained in detail in section 3.7, "Tool Messages" of the *OptoTerminal Programmer's Reference Manual*.

Since a non-drawable object is not visible on the display in Layout View, its handler methods for tool messages are small and straightforward. The code for the handlers is as follows:

```
#if _TOOL
'This function is called by Qlarity ->
'Foundry when you change the attachment ->
'of an object (i.e. in response to ->
'dragging it around the object tree).
'Note: the purpose of this function is to ->
'allow you to attach to another ->
'container; you should not change your ->
'parent property or call Tool_Persist ->
'until you receive a MSG_TOOL_ATTACHED. ->
'A MSG_TOOL_ATTACHED will be sent in ->
'response to an Attach API function call.
func ToolAttach (attachTo as objref)
   handles MSG_TOOL_ATTACH

   Attach (me, attachTo)
endfunc

'This function is called in response to ->
'calling Attach (primarily in a handler ->
'for MSG_TOOL_ATTACH, but possibly in ->
'other places as well_.
func ToolAttached (newParent as objref)
   handles MSG_TOOL_ATTACHED

   'Set our parent property to reflect ->
   'who we are attached to and save
   parent := newParent
   Tool_Persist (parent) 'Save parent
endfunc

'This function is called when the user ->
'created an instance of this template by ->
'selecting it from the object palette and ->
'dragging it in the layout view.
'The handler for MSG_INIT is not called ->
'automatically, and if you want it ->
'called, you should call it manually.
'You should call Tool_Persist on all the ->
'properties that you set up in this ->
'message handler.

func ToolDragCreate (parentObj as objref)
   handles MSG_TOOL_DRAGCREATE

   parent := parentObj
   Tool_Persist (parent)

   'Call our MSG_INIT handler
   Startup()
endfunc

#endif
```

First, note that all of the tool handlers are enclosed inside a "#if _TOOL/#endif" block. This excludes the tool handlers when the application is compiled for the OptoTerminal.

As noted in the comments, the MSG_TOOL_ATTACH message is sent to an object when it has been dragged onto a new parent in the Object Tree. An objref referencing the new parent is passed to the handler as a parameter. In most cases, the handler should merely call the Attach() API function to request attachment to the new parent. This is what the boilerplate code does.

If the object has restrictions on where it can be attached or what types of objects can serve as its parent, code should be added to the handler to enforce the restrictions. For example, the Tabs object in the Opto 22 Common object library can only attach to a Tab Container object. The MSG_TOOL_ATTACH handler for the Tabs object enforces this restriction.

After a requested attach is complete, the attached object receives a MSG_TOOL_ATTACHED message from Qlarity Foundry. An objref referencing the new parent is passed to the handler as a parameter. This provides an opportunity for the object to update its "parent" property (after the attach has been completed successfully). The new parent may be different from the parent that was passed to the Attach() API function in the MSG_TOOL_ATTACH handler.

Also, note that the boilerplate handler calls the Tool_Persist() API function after the parent property has been updated. The Tool_Persist() function should be called after any property update in a tool message handler.

The last handler included in the non-drawable boilerplate code is the MSG_TOOL_DRAGCREATE handler. When an object instance is created by selecting the template on the Object Palette and dragging it into the Layout View display, the object receives a MSG_TOOL_DRAGCREATE message from Qlarity Foundry. This allows the object to initialize itself properly. For a non-drawable object, the only parameter passed to the handler is an objref referencing the parent object. The handler updates its "parent" property and calls Tool_Persist(). Note that the MSG_INIT handler is called from the handler to complete the object initialization.

The tool message handlers for containers and area objects are identical. In addition to the handlers described above, the boilerplate code for these object types includes handlers for the MSG_TOOL_MOVE, MSG_TOOL_GETHANDLES, and MSG_TOOL_MOVEHANDLE messages.

Also, the MSG_TOOL_DRAGCREATE handler is larger and more complex than the handler for non-drawables.

The MSG_TOOL_ATTACH and MSG_TOOL_ATTACHED handlers are identical to the handlers in the non-drawable code, so they are omitted here. The code for the MSG_TOOL_MOVE handler is as follows:

```
'This function is called when the user ->
'drags an object instance with the mouse ->
'in Qlarity Foundry. dx and dy are ->
'relative offsets from the current ->
'location of the object.
func ToolMove (dx as integer, dy as integer)
   handles MSG_TOOL_MOVE

   xPos := xPos + dx
   yPos := yPos + dy
   Relocate (me, xPos, yPos)

   'Save the changes we just made
   Tool_Persist (xPos)
   Tool_Persist (yPos)
endfunc
```

When the user drags an object in Layout View, the object dragged receives MSG_TOOL_MOVE messages from Qlarity Foundry. The change in position is passed to the handler in the "dx" and "dy" values. The object updates its "xPos" and "yPos" properties to the new location and calls the Relocate() API to generate a MSG_DRAW that updates the screen (if necessary). Finally, the handler persists the values of the updated properties with calls to Tool_Persist().

The code for the MSG_TOOL_GETHANDLES handler is as follows:

```
'This function is called by Qlarity ->
'Foundry to obtain the coordinates of the ->
'resize grips. The arrays xCoords, ->
'yCoords, and cursors all contain 0 ->
'elements initially. This means that you ->
'should either call redim on those arrays ->
'and set their values, or declare local ->
'arrays, set the values of the local ->
'arrays, and then assign xCoords, yCoords ->
'and cursors to those local arrays. ->
'Closed indicates whether the Foundry ->
'should connect the first and last grips ->
'to make a closed object. xCoords, ->
'yCoords, and cursors should all contain ->
'the same number of elements when this ->
'function completes.
func ToolGetHandles (xCoords[] as ->
reference to integer,
       yCoords[] as reference to integer,
       cursors[] as reference to GuiCursors,
       closed as reference to boolean)
   handles MSG_TOOL_GETHANDLES

   dim csrs[8] as GuiCursors
   init csrs := [CSR_UPLEFT, CSR_UPDOWN, ->
   CSR_UPRIGHT, CSR_LEFTRIGHT, ->
   CSR_DOWNRIGHT, CSR_UPDOWN, ->
   CSR_DOWNLEFT, CSR_LEFTRIGHT]

   redim (xCoords, 8)
   redim (yCoords, 8)
   xCoords[0] = xPos
   xCoords[1] = xPos + width / 2 - 1
   xCoords[2] = xPos + width - 1
   xCoords[3] = xCoords[2]
   xCoords[4] = xCoords[2]
   xCoords[5] = xCoords[1]
   xCoords[6] = xPos
   xCoords[7] = xPos

   yCoords[0] = yPos
   yCoords[1] = yPos
   yCoords[2] = yPos
   yCoords[3] = yPos + height / 2 - 1
   yCoords[4] = yPos + height - 1
   yCoords[5] = yCoords[4]
   yCoords[6] = yCoords[4]
   yCoords[7] = yCoords[3]

   cursors = csrs
   closed = true
endfunc
```

The MSG_TOOL_GETHANDLES message is sent to an object by Qlarity Foundry when the object is created. All objects receive this message when a workspace is initialized. Qlarity Foundry requests the locations of the resize grips (sizing handles) that are drawn on an object when it is selected. The handler is passed references to three arrays (xCoords, yCoords, and cursors) and a boolean variable named "closed." The handler places values in these parameters to indicate the location and style of the resize grips.

The handler first creates and initializes an array of type GuiCursors. This is an enumerated type whose values indicate the style of the corresponding resize grip. The selected style determines what cursor is drawn when the mouse passes over the resize grip and the restrictions on drag movement for the resize grip. See section 3.7, "Tool Messages" in the *OptoTerminal Programmer's Reference Manual*.

The xCoords and yCoords arrays receive the horizontal and vertical locations for each resize grip, respectively. The size of the three arrays (xCoords, yCoords, and cursors) determines the number of resize grips that will be drawn. All three arrays should be sized to the same value. The boilerplate code sizes these arrays to 8, so 8 resize grips will be drawn for the object. The code then locates the grips on the perimeter of the object at locations calculated from the position and size properties.

Qlarity Foundry draws lines to connect each resize grip. The closed parameter determines whether a line connecting the first and last grips will be drawn. Ordinarily, this is desirable. However, certain objects, such as the Line and PolyLine objects in the Opto 22 Common object library, do not "enclose" a space on the screen and the last line is not desirable. A value of TRUE causes this line to be drawn. This property also determines whether clicking inside the object will select it in Layout View.

The code for the MSG_TOOL_MOVEHANDLE handler is as follows:

```
'This function is called in response to a ->
'user moving a resize grip within Qlarity ->
'Foundry. handleNum is the index into the ->
'arrays that were returned by the handler ->
'for MSG_TOOL_GETHANDLES. You should ->
'return true from this function.
func ToolMoveHandle (handleNum as ->
reference to integer, dx as integer, dy ->
as integer)
    handles MSG_TOOL_MOVEHANDLE

    dim newX, newY, newWidth, newHeight, ->
    tmp as integer
    newX := xPos
    newY := yPos
    newWidth := width
    newHeight := height

    if (handleNum == 0) then
        newX := xPos + dx
        newWidth := width - dx
        newY := yPos + dy
        newHeight := height - dy
    elseif (handleNum == 1) then
        newY := yPos + dy
        newHeight := height - dy
    elseif (handleNum == 2) then
        newY := yPos + dy
        newHeight := height - dy
        newWidth := width + dx
    elseif (handleNum == 3) then
        newWidth := width + dx
    elseif (handleNum == 4) then
        newWidth := width + dx
        newHeight := height + dy
    elseif (handleNum == 5) then
        newHeight := height + dy
    elseif (handleNum == 6) then
        newX := xPos + dx
        newWidth := width - dx
        newHeight := height + dy
    else
        newX := xPos + dx
        newWidth := width - dx
    endif

    if (newWidth < 1) then
        newX := newX + newWidth - 1
        newWidth := -newWidth + 2
        tmp = handleNum mod 4
        if tmp == 0 then
            handleNum = handleNum + 2
        elseif tmp == 2 then
            handleNum = handleNum -2
        else
            handleNum = (handleNum + 4) mod 8
        endif
    endif

    if (newHeight < 1) then
        newY := newY + newHeight - 1
        newHeight := - newHeight + 2
        tmp = handleNum mod 4
        if tmp == 0 then
            handleNum = (handleNum + 6) mod 8
        elseif tmp == 1 then
            handleNum = (handleNum + 4) mod 8
        else
            handleNum = (handleNum + 2) mod 8
        endif
    endif

    xPos := newX
    yPos := newY
    width := newWidth
    height := newHeight

    Relocate (me, xPos, yPos)
    Resize (me, width, height)

    'Save any changes we just made
    Tool_Persist (xPos)
    Tool_Persist (yPos)
    Tool_Persist (width)
    Tool_Persist (height)
endfunc
```

When the user drags a resize grip in Layout View, the object being resized receives a MSG_TOOL_MOVEHANDLE message from Qlarity Foundry. The grip being manipulated and the change in position are passed to the handler as parameters. The handler typically determines which grip is being dragged, calculates its new size and position, updates its properties, requests a redraw, and persists the values with Tool_Persist. This is what the boilerplate code does. There is also some validation code to prevent the object from being sized to a height or width of 0.

The "handleNum" parameter is passed to the handler as a reference, which allows the handler to change the currently selected grip if desired. Notice that the boilerplate code reassigns "handleNum" when the object is dragged "inside out" (i.e., a grip at the bottom is dragged above the top of the object, or a grip on the right edge of the object is dragged past its left edge).

Finally, the container/area object boilerplate code for the MSG_TOOL_DRAGCREATE handler is as follows:

```
'The following function is called when ->
'the user created an instance of this ->
'template by selecting it from the object ->
'palette and dragging it in the layout view.
'The handler for MSG_INIT is not called ->
'automatically, and if you want it ->
'called, you should call it manually.
'You should call Tool_Persist on all the ->
'properties that you set up in this ->
'message handler. (x1, y1) are the ->
'coordinates that the user started his ->
'drag and (x2, y2) are the coordinates ->
'that the mouse was released.
func ToolDragCreate (parentObj as objref, ->
x1 as integer, y1 as integer, x2 as ->
integer, y2 as integer)
    handles MSG_TOOL_DRAGCREATE

    if (x2 < x1) then
        xPos := x2
        width := x1 - x2 + 1
    else
        xPos := x1
        width := x2 - x1 + 1
    endif

    if (y2 < y1) then
        yPos := y2
        height := y1 - y2 + 1
    else
        yPos := y1
        height := y2 - y1 + 1
    endif

    parent := parentObj
    'Save the properties that we just set
    Tool_Persist(parent)
    Tool_Persist (xPos)
    Tool_Persist (yPos)
    Tool_Persist (width)
    Tool_Persist (height)

    'Call our MSG_INIT handler
    StartUp()
endfunc
```

Since the location and size of an area object/container is determined by the extent of the click-and-drag operation in Layout View, this handler is necessarily more complex than the handler in the non-drawable boilerplate code. The (x1,y1) and (x2,y2) coordinates that are passed to the handler as parameters indicate the locations of the initial click and release, respectively, at the end of the drag.

The handler calculates the position and size properties of the object based on the values of (x1,y1) and (x2,y2). It also sets the parent, persists the changed properties, and calls the MSG_INIT handler to complete the object initialization.

The boilerplate handlers should suffice for most objects, and the code is easily modified to suit unusual circumstances. Writing these handlers "from scratch" can be tedious and error-prone. For this reason, using the boilerplate code as a basis for new object template designs is recommended.

10.4.3 Adding Object Template Documentation

"AutoDoc" is a Qlarity Foundry feature that automatically documents the object templates included in a workspace. All libraries provided with Qlarity Foundry, as well as functions, variables, and APIs, are already included in Object Documentation and the optional "extra pop-up Help" (see section 6.2, "Layout").

Press **<F1>** or select **Object Documentation** on the Help menu to open Object Documentation.

To add documentation on global variables, global functions, objects, methods, or properties that you create, you must add the proper "#doc" code to your code. AutoDoc will then find the information you enter and add it to the Object Documentation.

To document a program item, type the following in the same location as the item (e.g., to document an object property, enter the documentation within the object):

```
'This documents an object property OR ->
'global variable
#doc property <name>
#doc prop <name>

'This documents a global function or ->
'object method
#doc func <name>
#doc function <name>
#doc override <name> '(only use this ->
'if the function is designed to be ->
'overridden)

'This documents an object itself
#doc object <name>
```

To enter a description of an item, begin each line of the description with a tilde (~):

```
#doc object buttonBDF
~The ButtonBDF object is a versatile ->
~object that responds to touchscreen ->
~input.
~When you touch an instance of the ->
~ButtonBDF object, the function called ->
~Click() is called.
```

The tilde must be the first non-whitespace character on the line, otherwise it is treated as a comment.

When documenting a function, you can add parameter documentation as follows:

```
#doc func CalcFactoral
#param toCalc:The function calculates the
factoral of this parameter.
~CalcFactoral calculates the factoral ->
~of its parameter. The factoral of n ->
~(written n!) is defined as n * (n-1) *
~(n-2) * ... 1.
func CalcFactoral (toCalc as integer) ->
returns integer
...
endfunc
```

Note that "#param" lines must be completed in one line. You cannot use the line continuation character to extend them to more than one line.

10.5 Guidelines for Designing New Object Templates

This section contains advice and collected wisdom regarding the design of new Qlarity object templates. Designing good object templates is the most complex task in Qlarity programming. Once this skill is mastered, the full potential of Qlarity can be utilized in your application development.

Look at template code in the Opto 22 libraries.
Each library contains the full source code for every object template in the library. You are encouraged to study the code in these libraries to learn the techniques and conventions used in their design. Often the easiest way to create a custom object is to modify a template that is similar to the desired object.

Use the boilerplate code.
This code provides a complete, working Qlarity object ready for customization. Do not unnecessarily "reinvent the wheel."

Use a consistent style.
This is important in any programming project. Opto 22 has developed a set of Qlarity programming style conventions for objects that are designed in-house. This was also used in the boilerplate code. The style guide is available on the Qlarity Web site: www.qlarity.com. Although you may wish to modify these guidelines or develop your own, use of a consistent style will make your object designs easier to develop, maintain, and use.

Use consistent naming conventions for properties and methods.
This is actually part of a consistent style. The Opto 22 Qlarity style guide lists the recommended names for properties that are commonly used in objects. Using these conventions aids the designer when the object is used in an application; it also increases readability of the object template source code.

Use validation functions.
A validation function is the link between an object's properties and its behavior. Almost every property should have a validation function so that the object can properly respond to property changes.

Become familiar with the API library.
All interaction with the Qlarity-based hardware is achieved through API function calls. You should be familiar with the functions available and their capabilities.

Provide default values for all properties.
This provides a predictable starting point for each newly created object instance.

Provide simple to use override functions.
Many of the objects in the Opto 22 libraries include simple override functions that are called from event handlers. Most of these functions are empty by default, but they provide a way for each object instance to determine how it will respond to a given event. These override functions often take no parameters and have no return value, so they are easy to remember and use. For example, many button-type objects have a Click() function that is called from the handler for a touch screen press and release. This function is empty by default, but it provides an interface for each object instance to customize its response to a press or release event.

Keep the MSG_DRAW handler small.
Drawing requires the most computations on the OptoTerminal. The MSG_DRAW handler for an object should be as concise and efficient as possible. Where convenient, data needed for drawing should be pre-calculated and stored in private properties for later use in the MSG_DRAW handler.

Use user messages to communicate between objects.
User messages provide an excellent way to communicate between objects, because the sending object does not require any knowledge about the receiving object. This feature allows Qlarity to be extended in a number of powerful ways.

10.6 Where to Go From Here

The full power and potential of the Qlarity programming language is now at your disposal. As your experience with Qlarity grows, you will likely create objects that might be useful for others. Opto 22 encourages Qlarity programmers to submit their object template designs to the Opto 22 Web site (www.opto22.com) where they can be shared with others.

Questions and feedback about the Qlarity programming language and the Opto 22 OptoTerminal should be sent via e-mail to support@Opto22.com. You should also check the www.opto22.com Web site occasionally for software and documentation updates, new library objects, and support materials.

APPENDIX A

GLOSSARY OF SOFTWARE TERMS

API Function
A function that is called from the user application to interact with the Qlarity-based system software (firmware) and hardware. You use API functions to draw to the screen, render text and bitmaps, send characters to the serial and network ports, enable/disable objects, manipulate object order in the hierarchy, perform complex math and so on.

Area Object
An object that directly interacts with the terminal display by drawing something on the display and/or processing area-based messages. Examples of area objects include: text objects, bitmap objects and line objects.

BFF File Format
Binary file format (BFF) required for a user application to run on a OptoTerminal. When a workspace is compiled, it is converted to a BFF format.

Container Object
An object that contains other objects (e.g, a form or screen). On the Qlarity-based display, a container may represent a portion of the display or all of the display. Whether or not a container is displayed depends on its position in the object hierarchy and whether or not containers/objects in front of the object are transparent. Also, the hierarchy determines the order for messaging.

Enabled and Disabled Objects
An object may be enabled or disabled without deleting it from the user application. An enabled object can process most messages. Disabled objects are not drawn on the display and are not eligible to process messages. In Qlarity Foundry, an object can be enabled or disabled using its object properties, but disabled objects continue to be displayed.

Event
An occurrence that signals a change in the terminal state, such as a touch screen press, a keyboard press, a serial character receive, or a system time tick. Events generate messages that allow the user application to react to the event in a defined way.

Globals
Code and data that exists independently of objects in an application.

Libraries
Collections of predefined object templates and/or resources available in Qlarity Foundry. Some libraries are provided by Opto 22; however, advanced users may create their own libraries.

Message
Delivers information about an event to the user application. User-defined functions and object methods are called when the message they handle is generated.

Message Handling System
System that handles event processing for the OptoTerminal. When an event occurs, the system software and/or hardware drivers generate a message indicating that the event has occurred. The message is passed through the message handling system in the Qlarity-based system software, which reviews the object hierarchy and determines which object gets the message and in what order the message will be processed.

In order to process a message, an object must be enabled (which also causes area and container objects with a defined area to be drawn on the screen). Disabled objects do not process most messages.

Methods
Functions contained in an object comprising much of the code portion of the object. The methods for each type of object are defined in the object template, but each object instance may override some of the object template's methods. Methods define the behavior of an object at runtime and typically either manipulate or depend on the values of the object's properties.

Method Override
A function in an object instance that "overrides" (or replaces) the method defined in the object template. Each object instance may override some of the template's methods. The override function has the same name and parameters as the template method, and the code in an override may call the template method if desired.

Non-Drawable Object
An object that has a purpose not directly related to the terminal display. Examples of non-drawable objects include: communication, event, and keyboard objects.

Object
The basic unit of a user application. You define objects by type with each type representing a display element, button or other function. *Properties* and *methods* define what an object is and how it behaves. "Object types" may take many forms and serve any number of purposes. Object types fall into three categories: container objects, area objects and non-drawable objects.

Object Hierarchy
For the display, it is the *Z-order*, or the order in which objects are layered. For messaging, it is the order in which objects are prioritized. The top object in the object hierarchy receives the highest messaging and display priority (i.e., it is "on top" of other objects). However, any enabled objects, regardless of their position in the hierarchy, may receive or send messages. You can manipulate the position of an object in the hierarchy with API functions. Each container (including the root container) maintains a list of objects attached to it (its children). The order of objects on this list indicates the Z-ordering of objects (from front to back). The Z-ordering may also be manipulated at runtime using the Z-order primitive API functions.

Object Instance
An occurrence of an object in a user application. An object instance is defined by the object template on which the object is based. Each object maintains its own properties and may contain code for method overrides.

Object Template
The programming code that defines an object. Each object instance in a user application is based on a template. Object templates can be a part of a workspace or supplied through one or more libraries. The template defines which properties and methods are included in the object.

Properties
Variables that are stored in an object and comprise the data portion of the object. The properties for each type of object are defined in the object template, but each object instance maintains its own properties. In Qlarity Foundry, object properties can be changed in Object Properties in Layout View without the need to modify the programming code. Properties may also be changed at runtime to reflect the current state of the object.

Qlarity
A BASIC-like scripting language that is used to write user applications for the OptoTerminal.

Resources
Bitmap images, fonts, audio files, and binary data files available for use in an application.

Root Container
A root-level container that uses the global properties and methods as its properties and methods. The root container is best viewed as an abstraction. It gives you a place to attach all other objects. An object that is not attached to a container object is attached to the root container by default. The root container is not a true object and cannot be disabled. Its area cannot be modified. The root container is a launching point for newly generated messages. This means global message handlers have the highest priority for receiving new messages.

User Application
A user-created program that controls the OptoTerminal. A user application interacts with the terminal's system software (firmware) to define and control the terminal's display, touch screen, speaker and input/output actions.

Validation Method
An object method that is implicitly called when the object property of the same name is assigned a value. Although the name validation implies that the function is used to validate the value before it is assigned to the property, the validation method may be used for any purpose. This is a powerful feature of Qlarity, because it allows object behavior to be controlled solely by manipulating the object properties.

Workspace
A file created in Qlarity Foundry that contains Qlarity programming code and data for a user application. You use Qlarity Foundry to compile the workspace file into a user application and to download the user application to the OptoTerminal. You can create any number of workspaces.

Z-Order
The order in which objects are layered. The order of objects in the Object Tree indicates the Z-ordering of objects (from front to back). The Z-ordering may also be manipulated at runtime using the Z-order primitive API functions. (Also see Object Hierarchy.)

APPENDIX B

EZOBJECTS LIBRARY

The EZObjects library (ezlib) contains object templates that have event functionality programmed into them. All EZObjects contain the programming needed to respond to an event (e.g., touch screen press, timer action, etc.). The response is a transfer of data from a source to a target object. You set up an event for an EZObject by entering or selecting the appropriate properties in Object Properties. No additional programming is required.

For example, you can set up an EZCircle or EZImage as a button that, when pressed on the screen, triggers an action such as changing text, switching screens on the terminal display, updating a simulated gauge, or sending data out the serial or Ethernet port.

If you want an object to respond to multiple events (e.g., screen press and then screen release), or if you want an event to send data to multiple objects, you must add additional objects called "EZEvent" objects. By linking one or more EZEvent objects to an EZObject, you can expand the EZObject's event processing capabilities. You link EZEvent objects to EZObjects using the "eventobj" property (explained later in this section). An EZEvent object cannot stand alone to define an event. EZEvents contain only event-related properties that are linked to an EZObject.

For example, you can set up an EZCircle as a button that, when pressed, sends a text string to a text object and also sends the text string (or other data) out the communications port.

NOTE: also see Event Builder
Qlarity Foundry's Event Builder allows you to add event/action functionality to any appropriate object. Refer to section 8.4, "Event Builder" for more information.

The following image shows an EZImage's object properties, including the properties required to set up an event.

(Name)	String	EZImage_1
enabled	boolean	true
parent	objref	default
xpos	integer	77
ypos	integer	48
transparent	boolean	false
transparentcolor	color	rgb_magenta
ezimage	bitmap	default
blink	boolean	false
blinkrate	integer	60
event	"imageev...	image_screen...
eventsourceobj	objref	scale_1
eventsourcepro...	string	""
eventtargetobj	objref	serial_1
eventtargetprop...	string	""
eventtargetvalue	string	""
eventobj	"ezevent"	empty

The following image shows an EZEvent's object properties.

(Name)	String	ezevent_1
enabled	boolean	true
parent	objref	default
event	"ezeventtyp...	eznothing
eventsourceobj	objref	default
eventsourceproperty	string	default
eventtargetobj	objref	default
eventtargetproperty	string	default
eventtargetvalue	string	default
eventobj	"ezevent"	empty

B.1 Set Up an Event with EZOBjects

To set up an event using EZOBjects, do the following.

1. Add the EZObject that will be used to perform the event (e.g., EZCircle, EZImage).

2. If you want the object to perform more than one event (e.g., a screen press and the screen release), or if you want the event to send data to multiple objects, add the applicable number of EZEvent objects.

For example, if you want to define screen press and screen release events, add the EZObject, then add one EZEvent object. You can define the screen press event in the EZObject properties and the screen release event in the EZEvent object properties, which you will link to the EZObject.

3. Define the following in the object properties of the EZObject and any EZEvent objects linked to it.

   - The event that will trigger the action (specified in the "event" property)
   - The source of the data to be sent using *one* of the following methods:
     - A source object and property (specified in the "eventsourceobj" and "eventsourceproperty" properties)
     - The data to be sent (typed in the "eventtargetvalue" property)
   - The target object and property to receive the data (e.g., text object, gauge object, serial object, etc.), specified in the "eventtargetobj" and "eventtargetproperty" properties

Following are the properties required to define an event.

eventobj
This property is only required if you are linking EZEvent objects to the EZObject. It allows an object to perform multiple actions from a single event or to handle more than one event. In the EZObject's property, click the drop-down list, and select the name of the EZEvent object; or, if using more than one EZEvent object, any one of the EZEvent objects to be linked.

If linking only one EZEvent object, you do not need to enter anything in this property for the EZEvent object.

If linking more than one EZEvent object, in the first EZEvent object, select the name of another EZEvent Object to be linked. In the next EZEvent object, select the name of another EZEvent Object to be linked; and so on, until each EZEvent object is linked to another. You do not need to enter anything in this property for the last linked EZEvent object. The order in which EZEvents are linked does not matter.

For example, if you are defining an event that requires five objects (one EZOjbect and four EZEvent objects), link them together as follows: In the "eventobj" property of the EZObject, select ezevent_1; in ezevent_1's property, select ezevent_2; in ezevent_2's property, select ezevent_3; in ezevent_3's property, select ezevent_4. When the event that triggers the action occurs, the message announcing the event is passed from the EZObject to ezevent_1, which processes its defined event (if appropriate) and passes the message to ezevent_2, and so on through each EZEvent object.

event
From the drop-down list, select the event that you want to trigger the action. Valid events are determined by your hardware configuration and may include:

- A touch screen press, press and move, or release
- A keyboard key press or release
- A change in a value (increment or decrement)
- A timer action

eventsourceobj
If the source of the data to be transferred is an object, select the source object from the drop-down list (you also need to select the "eventsourceproperty"). If the source is a specified value (rather than an object), enter the value in the "eventtargetvalue" property field.

eventsourceproperty
Type the name of the property that contains the source data (from the source object's properties). Be sure to type the property name exactly as it appears in Object Properties.

eventtargetobj
Select the target object for the action from the drop-down list. The target object is the object that will be changed when the event takes place, such as moving a gauge.

eventtargetproperty
Type the name of the target object's property that will be changed when the event takes place. Be sure to type the property name exactly as it appears in Object Properties.

eventtargetvalue
If you want a specific value transferred to the target property (rather than data from a source object) when the event takes place, enter it in this property. If you are entering an array, enclose it in square brackets and separate each number by commas (e.g., [10, 20, 30, 40, 50]).